```
# Check important enviroment variables
import os
print("JAVA_HOME =", os.environ.get("JAVA_HOME"))
print("SPARK_HOME =", os.environ.get("SPARK_HOME"))
print("PYSPARK_SUBMIT_ARGS =", os.environ.get("PYSPARK_SUBMIT_ARGS"))
```

```
JAVA_HOME = /usr/lib/jvm/temurin-11-jdk-amd64
SPARK_HOME = /usr/lib/spark
PYSPARK_SUBMIT_ARGS = --deploy-mode client pyspark-shell
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext
spark.version

from pyspark.sql.functions import (
    col, when, lead, input_file_name, regexp_extract, to_date,
    udf, sum as spark_sum, mean, stddev, min, max, count, sum
)
from pyspark.sql.window import Window

from pyspark.sql.types import (
    StructType, StructField, StringType, IntegerType, DoubleType, DateType, LongType
)

from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.pipeline import PipelineModel
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/15 19:56:35 INFO SparkEnv: Registering MapOutputTracker
25/05/15 19:56:35 INFO SparkEnv: Registering BlockManagerMaster
25/05/15 19:56:35 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
25/05/15 19:56:35 INFO SparkEnv: Registering OutputCommitCoordinator
```

Financial Article Sentiment Analysis

Student: Ashley Sun

On April 7th, Monday, a false headline about President Trump considering a 90-day pause on the tariffs led to a swift surge of the S&P 500, adding trillions in market value within minutes, then crashing down rapidly after the White House refuted the report. This incident highlights the need for a performant live sentiment analysis system to make trading decisions. For the project, I built a scalable ML pipeline using big data tools such as Google Cloud Data Ingestion, Hadoop, HDFS, Spark, and Spark MLlib. I trained my own financial news sentiment analysis model from scratch and tested the model against historical pricing data.

Tasks

- Ingest financial news, historical price data, and sentiment word lists at scale
- Clean, format, and efficiently store the input
- Featurize and label the articles
- Train and test our own sentiment analysis model on dated financial news
- Efficiently manage and store intermediate results such as models
- Combine sentiment prediction with historical financial data for evaluation
- Efficiently evaluate and backtest the strategy

Data Sources:

- Nasdaq Financial News: ~26GB, 15 millions dated financial articles
- Nasdaq Historical Prices: ~ 4700 csv files. Each file contains historical prices of a symbol.
- Loughran-McDonald Sentiment Word List: sentiment word list focusing on financial news
- Liu Bing Sentiment Word List: supplemental generic sentiment word list

Challenges and Solutions

- Storing and processing data at scale is difficult.
    - Solution: I utilized Google Data Ingestion and HDFS on Dataproc to manage all my data. I manually uploaded the data files to Ingestion portal, logged into Dataproc and copied it to HDFS.
- NYU Google Cloud Dataproc does not have many ML libraries such as pytorch, Scikit Learn, or SparkNLP.
    - Solution: I built my own ML model with very basic libraries such as SparkML.
- Our JupyterLab (the one we used for homework) is not on Dataproc and cannot handle the amount of data required.
    - Solution: I ran a Jupyter Notebook with PySpark on DataProc. However it is still quite challenging due to environmental setup.
        - Solution on running Jupyter Notebook on NYU Dataproc:

```
export JAVA_HOME=/usr/lib/jvm/temurin-11-jdk-amd64
export SPARK_HOME=/usr/lib/spark
export PYSPARK_PYTHON=python3
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
unset SPARK_SUBMIT_OPTIONS
unset PYSPARK_SUBMIT_ARGS
export PYSPARK_SUBMIT_ARGS="--deploy-mode client pyspark-shell"
jupyter notebook
```

        - To visit the notebook from my laptop, I installed GCloud and did a port forwarding:

```
gcloud compute ssh nyu-dataproc-m --project hpc-dataproc-19b8 --zone us-central1-f -- -N -L 8888:localhost:8
```

Sources: https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/cloud-computing/dataproc

Raw Data Processing

Raw artical data is downloaded from https://huggingface.co/datasets/Zihan1004/FNSPID/resolve/main/Stock_news/nasdaq_exteral_data.csv to my local laptop. Then I uploaded to https://dataproc.hpc.nyu.edu/ingest Data Ingestion Console. The data there is stored in Google Cloud Storage and it's temporary. To use the data in Spark in this notebook, I copied the data to hdfs with hadoop fs -cp gs://nyu-dataproc-hdfs-ingest/****/nasdaq_exteral_data.csv.

Even though it is a CSV file, the format is bad for Spark. Instead of 1 record per row, the Article column spans multiple line. Each article is inside of "..". This is bad for parallelizing task because in order to find the next record, we have to parse each line until a ". We cannot split the file arbitrariy since we might cut a record in half. As we see later, this dataset has **15,549,299** records. Proper preprocessing and storage are neccesary.

To address this, I'm loading the raw file once with proper parsing and storing the cleaned result in Spark Parquet format on HDFS. Parquet is a columnar storage format that is fast for Spark.

```
schema = StructType([
    StructField("Unnamed: 0", IntegerType(), True),        # likely an index column
    StructField("Date", DateType(), True),                 # e.g., 2023-05-01
    StructField("Article_title", StringType(), True),
    StructField("Stock_symbol", StringType(), True),
    StructField("Url", StringType(), True),
    StructField("Publisher", StringType(), True),
    StructField("Author", StringType(), True),
    StructField("Article", StringType(), True),
    StructField("Lsa_summary", StringType(), True),
    StructField("Luhn_summary", StringType(), True),
    StructField("Textrank_summary", StringType(), True),
    StructField("Lexrank_summary", StringType(), True),
])

# Intentionally commented out don't run this again

# # The Aritcl fields are all multi-line
# df = spark.read.csv(
#     '/user/***/nasdaq_exteral_data.csv',
#     header=True,
#     multiLine=True,
#     escape='"',
#     quote='"',
#     mode='DROPMALFORMED'
# )

# # Convert once to parquet so we can read it out faster in the future
# df.select("Date", "Article_title", "Stock_symbol", "Article").repartition(8).write.mode("overwrite").parquet('/user/****/nasda
```

```
# Load the data from pre-processed parquet saved on my HDFS
df = spark.read.parquet('/user/****/nasdaq_parquet/')
print(df.count())
df.limit(10).toPandas()
```

```
15549299
```

	Date	Article_title	Stock_symbol	Article
0	2007-01-11 00:26:00 UTC	ANALYSIS-US Mideast peace bid faces many obsta...	None	None
1	2014-12-03 00:00:00 UTC	Equifax Canada Reports Consumer Debt Grows to ...	EFX	None
2	2016-12-01 00:00:00 UTC	Силуанов назвал главную задачу обновленной нал...	None	Налоговая система России после 2018 года должн...
3	2023-10-09 00:00:00 UTC	IJR, ONTO, FN, RMBS: ETF Outflow Alert	GSEE	Looking today at week-over-week shares outstan...
4	2017-12-06 00:00:00 UTC	Американские учителя рассказали о неумеющих чи...	None	Учителя и учащиеся некоторых школ в Калифорнии...
5	2021-08-11 00:00:00 UTC	After Hours Most Active for Aug 11, 2021 : CLO...	HBAN	The NASDAQ 100 After Hours Indicator is down -...
	2016-01-09 03:54:00			

Load Sentiment Words for model training

In order to train my own model, I need to know what words are positive and what are negative. Instead of enumerating it myself, here is a well-known dataset for determining the sentiment of a word. This dataset is geared towards financial articles. https://sraf.nd.edu/loughranmcdonald-master-dictionary/. This is also uploaded to data injest then I copied it to HDFS. This file is 86k lines.

```
# Load csv from hdfs
lm_df = spark.read.csv(
    "/user/****/Loughran-McDonald_MasterDictionary_1993-2024.csv",
    header=True,
    inferSchema=True
)
```

```
lm_df.printSchema()
print(lm_df.count()) # This is small so we can do this
lm_df.limit(10).toPandas()
```

```
root
 |-- Word: string (nullable = true)
 |-- Seq_num: integer (nullable = true)
 |-- Word Count: integer (nullable = true)
 |-- Word Proportion: double (nullable = true)
 |-- Average Proportion: double (nullable = true)
 |-- Std Dev: double (nullable = true)
 |-- Doc Count: integer (nullable = true)
 |-- Negative: integer (nullable = true)
 |-- Positive: integer (nullable = true)
 |-- Uncertainty: integer (nullable = true)
 |-- Litigious: integer (nullable = true)
 |-- Strong_Modal: integer (nullable = true)
 |-- Weak_Modal: integer (nullable = true)
 |-- Constraining: integer (nullable = true)
 |-- Complexity: integer (nullable = true)
 |-- Syllables: integer (nullable = true)
 |-- Source: string (nullable = true)

86553
```

	Word	Seq_num	Word Count	Word Proportion	Average Proportion	Std Dev	Doc Count	Negative	Positive	Uncertainty	Litigious	Strong_
0	AARDVARK	1	755	2.955070e-08	1.945421e-08	4.078069e-06	140	0	0	0	0	
1	AARDVARKS	2	3	1.174200e-10	8.060019e-12	8.919011e-09	1	0	0	0	0	
2	ABACI	3	9	3.522600e-10	1.089343e-10	5.105359e-08	7	0	0	0	0	
3	ABACK	4	29	1.135060e-09	6.197922e-10	1.539279e-07	28	0	0	0	0	
4	ABACUS	5	9620	3.765268e-07	3.825261e-07	3.421836e-05	1295	0	0	0	0	
5	ABACUSES	6	0	0.000000e+00	0.000000e+00	0.000000e+00	0	0	0	0	0	
6	ABAFT	7	4	1.565600e-10	2.144787e-11	2.373367e-08	1	0	0	0	0	
7	ABALONE	8	149	5.831860e-09	4.729504e-09	1.031859e-06	52	0	0	0	0	
8	ABALONES	9	1	3.914000e-11	7.715206e-11	8.537449e-08	1	0	0	0	0	
9	ABANDON	10	154158	6.033745e-06	4.824004e-06	3.261271e-05	76324	2009	0	0	0	

```
# Keep only relevant columns
basic_cols = [
    "Word",
    "Word Proportion",
    "Average Proportion",
]
category_cols = [
    "Negative",
    "Positive",
    "Uncertainty",
    "Litigious",
    "Strong_Modal",
    "Weak_Modal",
    "Constraining"
]
selected_cols = basic_cols + category_cols

lm_df_filtered = lm_df.select(*selected_cols)
```

Converting from versioned category value to True and False

From the website, it states: "The sentiment categories are: negative, positive, uncertainty, litigious, strong modal, weak modal, and constraining. The sentiment words are flagged with a number indicating the year in which they were added to the list. Note: A year preceded by a negative sign indicates the year/version when the word was removed from the sentiment category."

The end goal is to have a list of words for each category, so first we convert the versioned to true false

```
category_cols = [
    "Negative",
    "Positive",
    "Uncertainty",
    "Litigious",
    "Strong_Modal",
    "Weak_Modal",
    "Constraining"
]

# For integer-type version columns
for col_name in category_cols:
    lm_df_filtered = lm_df_filtered.withColumn(
        col_name,
        when(col(col_name).isNotNull() & (col(col_name) > 0), True).otherwise(False)
    )


neg_words = lm_df_filtered.filter((col("Negative") == True) | \
                                  (col("Litigious") == True) | \
                                  (col("Constraining") == True) ).select("Word").rdd.flatMap(lambda x: x).collect()
neg_regex = "(?i)\\b(" + "|".join([w.lower() for w in neg_words]) + ")\\b"
neg_set = set(neg_words)
neg_size = len(neg_set)
print(f'Size of neg word set: {neg_size}')


pos_words = lm_df_filtered.filter((col("Positive") == True) | \
                                  (col("Strong_Modal") == True) | \
                                  (col("Weak_Modal") == True)).select("Word").rdd.flatMap(lambda x: x).collect()

pos_regex = "(?i)\\b(" + "|".join([w.lower() for w in pos_words]) + ")\\b"
pos_set = set(pos_words)
pos_size = len(pos_set)
print(f'Size of pos word set: {pos_size}')
```

```
Size of neg word set: 3245
Size of pos word set: 389
```

Adding Liu Bing Word Sets for Balancing

Noticed that this opinion word list is very imbalanced. I found another list called https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html which contains more generic positive and negative words.

```
# Load and filter lines (skip empty and comment lines)
pos_rdd = sc.textFile("/user/****/positive-words.txt") \
            .filter(lambda line: line.strip() and not line.strip().startswith(";")) \
            .map(lambda word: word.strip().upper())

neg_rdd = sc.textFile("/user/****/negative-words.txt") \
            .filter(lambda line: line.strip() and not line.strip().startswith(";")) \
            .map(lambda word: word.strip().upper())


# Example usage
pos_set_2 = set(pos_rdd.collect())
neg_set_2 = set(neg_rdd.collect())

print(f"Loaded {len(pos_set_2)} positive and {len(neg_set_2)} negative words.")
```

```
Loaded 2006 positive and 4783 negative words.
```

```
# Combine both sets
pos_set = pos_set.union(pos_set_2)
neg_set = neg_set.union(neg_set_2)
pos_size = len(pos_set)
neg_size = len(neg_set)
print(f'Size of pos word set: {pos_size}')
print(f'Size of neg word set: {neg_size}')
```

```
Size of pos word set: 2204
Size of neg word set: 7167
```

Limited Sample Size Test

We will first test our pipelines with a small sample size of 10k articles. In this debugging pipeline, I'm calling show() a lot which triggers evaluation. This is slow and impractical on large sample size.

```
# Reduce sample size and drop empty
df = spark.read.parquet('/user/****/nasdaq_parquet/')

# Limit size and filter empty,null articles
df_filtered = (
    df.limit(10000)
      .dropna(subset=["Article", "Stock_symbol"])  # removes nulls
      .filter(
          (col("Article").isNotNull()) & (col("Article") != "") &
          (col("Stock_symbol").isNotNull()) & (col("Stock_symbol") != "")
      )
)
```

Labeling the data

I first tried to use Spark regex to match against my word lists, but this doesn't work if both word lists are long. When an article matches something in both word lists, the pos always wins.

```
# This doesn't work when there are a lot of words in each category
# df_labeled = df_filtered.withColumn(
#     "label",
#     when(col("Article").rlike(pos_regex), 1)
#     .when(col("Article").rlike(neg_regex), 0)
#     .otherwise(None)
# ).filter(col("label").isNotNull())  # Keep only labeled rows
```

Therefore I am writing my own udf to compute a score base on count. Since the 2 sets of words have drastically different counts, I'm normalizing by their respective total count.

```
def score_by_count(text):
    import builtins
    if text is None:
        return 0
    words = text.lower().split()
    # usinging builtins.sum because I imported PySpark sum at the top
    pos_count = builtins.sum(1 for w in words if w.upper() in pos_set)
    neg_count = builtins.sum(1 for w in words if w.upper() in neg_set)
    pos_score = pos_count / pos_size
    neg_score = neg_count / neg_size
    return pos_score - neg_score

score_udf = udf(score_by_count, DoubleType())


df_filtered = df_filtered.withColumn("scaled_sentiment_score", score_udf(col("Article_title"))) \
       .withColumn("label", when(col("scaled_sentiment_score") > 0, 2)
                           .when(col("scaled_sentiment_score") == 0, 1)
                           .when(col("scaled_sentiment_score") < 0, 0)
                           .otherwise(None))


#Inspect some non-zero ones
df_filtered.filter(col("scaled_sentiment_score") != 0.0).select("Article_title", "scaled_sentiment_score").show(10)
```

```
+--------------------+----------------------+
|       Article_title|scaled_sentiment_score|
+--------------------+----------------------+
|Verizon Is Growin...|  -1.39528394028184...|
|New Media Investm...|  4.537205081669691...|
|Tech Today: Broad...|  4.537205081669691...|
|Atlantica Sustain...|  4.537205081669691...|
|Is SPDR MSCI EAFE...|  9.074410163339383E-4|
|Rio Tinto (RIO) I...|  4.537205081669691...|
|Silver Weekly Pri...|  -2.79056788056369...|
|Wall Street drops...|  -1.39528394028184...|
|Why Hecla Mining ...|  3.141921141387844E-4|
|Natural Gas Price...|  4.537205081669691...|
+--------------------+----------------------+
only showing top 10 rows
```

Article Featurization with TF-IDF

We use Term Frequency - Inverse Document Frequence to featurize each Article. It transforms each Article to a feature (numeric) vector. The ith value in the vector is the TF-IDF score of the ith word in the document, which signifies the importance of the word. TF-IDF has 2 distinct steps:

1. Term Frequency: TF(term, doc) = count of term in doc / total terms in doc: this is the frequency of the term within the current article. This represents how important a word is in a document. This step is easily parallelizable. Each Article can be proccessed independently (on different Spark nodes).
2. Inverse Document Frequency: IDF(term) = log(total number of articles / (1 + number of articles containing term). This step downweight words that are frequent across all documents such as "the", and "a" since they provide very little value.

Ideally, our feature vector is long enough so that every word in our sample set can be accounted for. In practice, we limit our vector size to numFeatures. We hash each word to get the corresponding index.

Logistic Regression on Categorical Label

In the previous steps, we labeld each article 0,1,2 (negative, neutral, positive) depending on the sentiment score and we created feature vector with TF-IDF. The final step is to choose a model. I'm using Logistic Regression (https://spark.apache.org/docs/latest//ml-classification-regression.html#logistic-regression) which is a very common supervised machine learning algorithm for classification. Specifically, I'm using Multinomial Logistic Regression since I have 3 categories.

```
# Tokenize the articles (like homework assignment)
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.Tokenizer.html
tokenizer = Tokenizer(inputCol="Article", outputCol="words")

# Remove stop words like "the", "is", "and", "a", "in", "to", "for", "on", "of", "with"
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StopWordsRemover.html
remover = StopWordsRemover(inputCol="words", outputCol="filtered")

# TF discussed above
tf = HashingTF(inputCol="filtered", outputCol="rawFeatures", numFeatures=200)

# IDF discussed above
idf = IDF(inputCol="rawFeatures", outputCol="features")

# Logistic Multi-label regression
lr = LogisticRegression(featuresCol="features", labelCol="label", family="multinomial", maxIter=100)

pipeline = Pipeline(stages=[tokenizer, remover, tf, idf, lr])


model = pipeline.fit(df_filtered)
# Always save it
model.write().overwrite().save("/user/****/sentiment_model")
```

```
# model = PipelineModel.load("/user/****/sentiment_model")

df_with_prediction = model.transform(df_filtered)

df_with_prediction.select("Article_title", "prediction", "probability").show(10, truncate=False)
```

```
+-------------------------------------------------------------------------------------------------------------+----------+----
|Article_title                                                                                                |prediction|prob
+-------------------------------------------------------------------------------------------------------------+----------+----
|IJR, ONTO, FN, RMBS: ETF Outflow Alert                                                                       |1.0       |[0.0
|After Hours Most Active for Aug 11, 2021 : CLOV, HBAN, CVE, QQQ, ACWI, CLF, MDLZ, KBWB, MTG, MFC, ABBV, LXP|1.0       |[0.0
|Weekly Economic Overview (September 3 - 6, 2013).                                                            |1.0       |[0.0
|Why Arcimoto Stock Skyrocketed 721.7% in 2020                                                                |1.0       |[0.1
|TJX Companies, Inc. (TJX) Ex-Dividend Date Scheduled for November 10, 2014                                   |1.0       |[0.0
|Notable ETF Inflow Detected - IGV, ADBE, CRM, INTU                                                           |1.0       |[0.0
|Up 50% in the Past Month, Is Beyond Meat Stock a Buy?                                                        |1.0       |[0.0
|Hercules Technology (HTGC) Q4 Earnings and Revenues Beat Estimates                                           |1.0       |[0.0
|CONSOL Lags EPS, Issues Guidance - Analyst Blog                                                              |1.0       |[0.0
```

```
|Noteworthy Wednesday Option Activity: WBA, CREE, ITCI                                                          |2.0       |[0.0
+---------------------------------------------------------------------------------------------------------------+----------+----
only showing top 10 rows
```

Testing with Price Data

In order to evaluat our sentiment analysis, we will load historical daily price data for the symbols.

I have downloaded price data from https://github.com/Zdong104/FNSPID_Financial_News_Dataset?tab=readme-ov-file and injested into HDFS. It is a folder containing **7694 price files of thousands of rows**. Each file is named by the stock symbol like below. After inspection I will load all csv files at once with manual schema because it is faster.

```
df_price = spark.read.csv("hdfs:///user/****/full_history/AAPL.csv", header=True, inferSchema=True)
df_price.show(5)
df_price.printSchema()
```

```
+----------+------------------+------------------+------------------+------------------+------------------+--------+
|      date|              open|              high|               low|             close|         adj close|  volume|
+----------+------------------+------------------+------------------+------------------+------------------+--------+
|2023-12-28|194.13999938964844| 194.6600036621093| 193.1699981689453| 193.5800018310547| 193.5800018310547|34014500|
|2023-12-27| 192.4900054931641|             193.5|191.08999633789065| 193.1499938964844| 193.1499938964844|48087700|
|2023-12-26| 193.6100006103516|193.88999938964844| 192.8300018310547| 193.0500030517578| 193.0500030517578|28919300|
|2023-12-22|195.17999267578125| 195.4100036621093|192.97000122070312| 193.6000061035156| 193.6000061035156|37122800|
|2023-12-21| 196.1000061035156| 197.0800018310547|             193.5|194.67999267578125|194.67999267578125|46482500|
+----------+------------------+------------------+------------------+------------------+------------------+--------+
only showing top 5 rows

root
 |-- date: date (nullable = true)
 |-- open: double (nullable = true)
 |-- high: double (nullable = true)
 |-- low: double (nullable = true)
 |-- close: double (nullable = true)
 |-- adj close: double (nullable = true)
 |-- volume: integer (nullable = true)
```

```
schema = StructType([
    StructField("Date", StringType(), True),
    StructField("Open", DoubleType(), True),
    StructField("High", DoubleType(), True),
    StructField("Low", DoubleType(), True),
    StructField("Close", DoubleType(), True),
    StructField("Adj Close", DoubleType(), True),
    StructField("Volume", LongType(), True)
])

# Read all the csvs, they will be concat together
price_df = spark.read.csv("/user/****/full_history/*.csv", header=True, schema=schema)

price_df = price_df.withColumn("Date", to_date("Date", "yyyy-MM-dd"))

# https://stackoverflow.com/questions/68915138/spark-sql-regex-to-extract-date-file-name-and-brand
price_df = price_df.withColumn(
    "Stock_symbol",
    regexp_extract(input_file_name(), r"([^/]+)\.csv$", 1)  # grabs the symbol from path
)

# Select only needed columns
price_df = price_df.select("Date", "Stock_symbol", col("Adj Close").alias("adj_close"))
```

To test our sentiment prediction, we will take a 1 day long/short position on the selected symbol if the prediction is 2/0 respectively. To compute the return, we will compute future 1 day return: (Price(t+1)-Price(t))/Price(t).

```
window_spec = Window.partitionBy("Stock_symbol").orderBy("Date")

# Shift the price by 1
price_df = price_df.withColumn(
    "adj_close_plus_1", lead("adj_close", 1).over(window_spec)
)
```

```
# Compute the return
price_df = price_df.withColumn(
    "return",
    (col("adj_close_plus_1") - col("adj_close")) / col("adj_close")
)
```

Now we take a look at the tables. We want to join them on Stock_symbol and Date. For each artical, we will have the forward 1 day return corresponding to the symbol and date.

```
# Two table that we are joining
price_df.show(10)

# This show triggers a big evaluation
df_with_prediction.limit(10).select("Date","Stock_symbol","Article_title","prediction").show()
```

```
+----------+------------+------------------+------------------+--------------------+
|      Date|Stock_symbol|         adj_close|  adj_close_plus_1|              return|
+----------+------------+------------------+------------------+--------------------+
|1962-01-02|          AA|1.5366575717926023|1.5602117776870728|0.015328207355262016|
|1962-01-03|          AA|1.5602117776870728|1.5602117776870728|                 0.0|
|1962-01-04|          AA|1.5602117776870728|1.5583264827728271|-0.00120835834032...|
|1962-01-05|          AA|1.5583264827728271|1.5074503421783447|-0.03264793427880103|
|1962-01-08|          AA|1.5074503421783447|1.4952024221420288|-0.00812492437967...|
|1962-01-09|          AA|1.4952024221420288|1.4970871210098269|0.001260497468361...|
|1962-01-10|          AA|1.4970871210098269|1.4914336204528809|-0.00377633370670...|
|1962-01-11|          AA|1.4914336204528809|1.4603431224822998|-0.02084604875753054|
|1962-01-12|          AA|1.4603431224822998| 1.424540400505066|-0.02451665052277...|
|1962-01-15|          AA| 1.424540400505066|1.4132344722747805|-0.00793654446464...|
+----------+------------+------------------+------------------+--------------------+
only showing top 10 rows

+--------------------+------------+--------------------+----------+
|                Date|Stock_symbol|       Article_title|prediction|
+--------------------+------------+--------------------+----------+
|2023-12-13 00:00:...|       OXSQZ|Lantheus Holdings...|       1.0|
|2018-01-12 00:00:...|         WDC|Zacks.com feature...|       2.0|
|2017-08-22 00:00:...|         PSA|Public Storage's ...|       1.0|
|2023-12-15 00:00:...|       PAVMZ|Celanese (CE) Ral...|       1.0|
|2021-03-26 00:00:...|          AA|Notable Friday Op...|       2.0|
|2022-05-30 00:00:...|         ADM|Should iShares Ru...|       2.0|
|2023-10-19 00:00:...|        ESEB|READ: Rosenbluth ...|       1.0|
|2016-06-17 00:00:...|         PEP|New Philadelphia ...|       2.0|
|2022-02-07 00:00:...|        NDSN|Graham (GHM) Repo...|       1.0|
|2010-12-13 00:00:...|        ZUMZ|This Quarter's To...|       2.0|
+--------------------+------------+--------------------+----------+
```

```
# Join on symbol and date
joined_df = df_with_prediction.join(
    price_df,
    on=["Date", "Stock_symbol"],
    how="left"
)
```

```
joined_df.filter(col("return").isNotNull()).select("Date", "Stock_symbol", "Article_title", "prediction", "return").show(10)
```

```
+--------------------+------------+--------------------+----------+--------------------+
|                Date|Stock_symbol|       Article_title|prediction|              return|
+--------------------+------------+--------------------+----------+--------------------+
|2022-05-10 00:00:...|          AA|Should iShares Co...|       2.0|0.014684934419002848|
|2023-02-09 00:00:...|         ACB|Aurora Cannabis I...|       2.0|-0.01086955465418...|
|2021-08-31 00:00:...|         ACI|Noteworthy ETF In...|       1.0|-0.00592886368765...|
|2023-11-29 00:00:...|         ACT|Everest Group (EG...|       2.0|0.004349363219322416|
|2012-09-12 00:00:...|        ACTG|Zacks #1 Rank Add...|       1.0| 0.02628707597191735|
|2015-06-24 00:00:...|         AGI|Monsanto Tops Q3 ...|       1.0|-0.01733085329818...|
|2023-08-04 00:00:...|        AGIO|Agios (AGIO) Q2 E...|       0.0|0.003534962915327802|
|2023-12-07 00:00:...|       AGM-A|Why Is Inter Parf...|       1.0|                 0.0|
|2023-12-11 00:00:...|       AGM-A|2 Unstoppable Gro...|       1.0|                 0.0|
|2023-12-15 00:00:...|       AGM-A|Carnival Stock Ha...|       1.0|-0.01716141320192...|
+--------------------+------------+--------------------+----------+--------------------+
only showing top 10 rows
```

```
df_with_strategy_return = joined_df.filter(col("return").isNotNull())\
                                   .withColumn(
                                        "strategy_return",
                                        when(col("prediction") == 2.0, col("return"))
                                        .when(col("prediction") == 0.0, -col("return"))
                                        .otherwise(0.0)
                                    )
df_with_strategy_return.select("Date", "Stock_symbol", "Article_title", "prediction", "return", "strategy_return").show(10)
```

```
+--------------------+------------+--------------------+----------+--------------------+--------------------+
|                Date|Stock_symbol|       Article_title|prediction|              return|     strategy_return|
+--------------------+------------+--------------------+----------+--------------------+--------------------+
|2018-10-31 00:00:...|         ACB|It Took Just 1 We...|       0.0|-0.01029407296076...|0.010294072960761444|
|2018-11-27 00:00:...|         ACB|Now That the Pot ...|       2.0| 0.08363631277373343| 0.08363631277373343|
|2019-06-04 00:00:...|         ACB|Absent Immediate ...|       1.0|-0.01778907519848998|                 0.0|
|2019-08-28 00:00:...|         ACB|Despite Pot Stock...|       1.0|-0.00537635327469...|                 0.0|
|2019-11-12 00:00:...|         ACM|AECOM Technology ...|       1.0|-0.02018090743603...|                 0.0|
|2022-04-19 00:00:...|        ACMR|Validea Peter Lyn...|       2.0|-0.01525822857467...|-0.01525822857467...|
|2016-03-23 00:00:...|         ADI|Analog Devices to...|       2.0|0.005314507015057924|0.005314507015057924|
|2017-07-07 00:00:...|         ADI|Microsemi Rating ...|       2.0|0.003313426435224797|0.003313426435224797|
|2016-06-07 00:00:...|         ADP|Automatic Data Pr...|       1.0|0.010192086984231952|                 0.0|
|2013-12-18 00:00:...|        ADUS|Tenet Downgraded ...|       1.0|-0.03350292666390068|                 0.0|
+--------------------+------------+--------------------+----------+--------------------+--------------------+
only showing top 10 rows
```

```
stats = df_with_strategy_return.select(
    count("strategy_return").alias("n"),
    mean("strategy_return").alias("mean_return"),
    stddev("strategy_return").alias("std_return"),
    min("strategy_return").alias("min_return"),
    max("strategy_return").alias("max_return"),
    sum("strategy_return").alias("total_return"),
    sum(when(col("prediction") == 2.0, 1).otherwise(0)).alias("long_counts"),
    sum(when(col("prediction") == 1.0, 1).otherwise(0)).alias("neutral_counts"),
    sum(when(col("prediction") == 0.0, 1).otherwise(0)).alias("short_counts")
)

stats.show()
```

```
+----+--------------------+--------------------+--------------------+------------------+------------------+-----------+-----
|   n|         mean_return|          std_return|          min_return|        max_return|      total_return|long_counts|neutr
+----+--------------------+--------------------+--------------------+------------------+------------------+-----------+-----
|1382|5.250920769828881E-4|0.025672836615400586|-0.17098445749610666|0.7562499940395355|0.7256772503903514|        366|
+----+--------------------+--------------------+--------------------+------------------+------------------+-----------+-----

Approx. Median strategy return: 0.0
```

Results

number of articles = 10k
n (number of trades) = 1382
mean (average return) = 5.25e-4
stdev of return = 0.03
min return = -0.17
max max = 0.75
sum(return) = 0.75

The average return is near 0 but positive meaning we have a tiny edge. However, we get near 0 if we normalize the return by standard deviation. This is the Sharpe ratio, and it is saying that our return per unit risk is very low.

The sum of the return is 75% meaning that if we invest a fix dollar amount on each prediction, our cumulative return is 75%.

The results indicate that the data pipeline is working. We can proceed to train a full-size model.

Training Bigger Sample Size and Seperate Test Set

Steps are the same as above without intermediate triggering action. I am using all 15M articles instead of 10k in the example above. After the initial filter, we have rouhgly 2M. I am also splitting training and testing set in a 80-20 split.

```
# Reduce sample size and drop empty
df = spark.read.parquet('/user/****/nasdaq_parquet/')

# Train-test split
```

```
train_df, test_df = df.randomSplit([0.8, 0.2])

# Filtering Training
train_df_filtered = (
    train_df.dropna(subset=["Article", "Stock_symbol"])  # removes nulls
      .filter(
          (col("Article").isNotNull()) & (col("Article") != "") &
          (col("Stock_symbol").isNotNull()) & (col("Stock_symbol") != "")
      )
)

# Filtering Test
test_df_filtered = (
    test_df.dropna(subset=["Article", "Stock_symbol"])  # removes nulls
      .filter(
          (col("Article").isNotNull()) & (col("Article") != "") &
          (col("Stock_symbol").isNotNull()) & (col("Stock_symbol") != "")
      )
)

# Count just 1 column, it's columnar storage so this is faster but still takes a while
print(train_df_filtered.select("Date").count())
print(test_df_filtered.select("Date").count())

train_df_filtered = train_df_filtered.withColumn("scaled_sentiment_score", score_udf(col("Article_title"))) \
                                     .withColumn("label", when(col("scaled_sentiment_score") > 0, 2)
                                                         .when(col("scaled_sentiment_score") == 0, 1)
                                                         .when(col("scaled_sentiment_score") < 0, 0)
                                                         .otherwise(None))
```

```
1993258
498520
```

```
# Labeling


# Featurizing Article, all discussed above
tokenizer = Tokenizer(inputCol="Article", outputCol="words")
remover = StopWordsRemover(inputCol="words", outputCol="filtered")
tf = HashingTF(inputCol="filtered", outputCol="rawFeatures", numFeatures=200)
idf = IDF(inputCol="rawFeatures", outputCol="features")
lr = LogisticRegression(featuresCol="features", labelCol="label", family="multinomial", maxIter=100)
pipeline = Pipeline(stages=[tokenizer, remover, tf, idf, lr])

# Fitting
model = pipeline.fit(train_df_filtered)
# Always save it
model.write().overwrite().save("/user/****/sentiment_model_big")
```

```
# Evaluation on Train set
train_df_with_prediction = model.transform(train_df_filtered)

train_joined_df = train_df_with_prediction.join(
    price_df,
    on=["Date", "Stock_symbol"],
    how="left"
)

train_df_with_strategy_return = train_joined_df.filter(col("return").isNotNull())\
                                         .withColumn(
                                                "strategy_return",
                                                when(col("prediction") == 2.0, col("return"))
                                                .when(col("prediction") == 0.0, -col("return"))
                                                .otherwise(0.0))

stats = train_df_with_strategy_return.select(
    count("strategy_return").alias("n"),
    mean("strategy_return").alias("mean_return"),
    stddev("strategy_return").alias("std_return"),
    min("strategy_return").alias("min_return"),
    max("strategy_return").alias("max_return"),
    sum("strategy_return").alias("total_return"),
    sum(when(col("prediction") == 2.0, 1).otherwise(0)).alias("long_counts"),
    sum(when(col("prediction") == 1.0, 1).otherwise(0)).alias("neutral_counts"),
    sum(when(col("prediction") == 0.0, 1).otherwise(0)).alias("short_counts")
)

stats.toPandas()
```

```
         n  mean_return  std_return  min_return  max_return  total_return  long_counts  neutral_counts  short_counts
0  1636617     0.000148    0.205479   -1.348358  259.095241    242.897535       306328         1316561         13728
```

```
# Evaluation on Test set
model = PipelineModel.load("/user/****/sentiment_model_big")

test_df_with_prediction = model.transform(test_df_filtered)

test_joined_df = test_df_with_prediction.join(
    price_df,
    on=["Date", "Stock_symbol"],
    how="left"
)

test_df_with_strategy_return = test_joined_df.filter(col("return").isNotNull())\
                                        .withColumn(
                                               "strategy_return",
                                               when(col("prediction") == 2.0, col("return"))
                                               .when(col("prediction") == 0.0, -col("return"))
                                               .otherwise(0.0))

stats = test_df_with_strategy_return.select(
    count("strategy_return").alias("n"),
    mean("strategy_return").alias("mean_return"),
    stddev("strategy_return").alias("std_return"),
    min("strategy_return").alias("min_return"),
    max("strategy_return").alias("max_return"),
    spark_sum("strategy_return").alias("total_return"),

    # Prediction counts
    spark_sum(when(col("prediction") == 2.0, 1).otherwise(0)).alias("long_counts"),
    spark_sum(when(col("prediction") == 1.0, 1).otherwise(0)).alias("neutral_counts"),
    spark_sum(when(col("prediction") == 0.0, 1).otherwise(0)).alias("short_counts"),

    # Long success/failure
    spark_sum(when((col("prediction") == 2.0) & (col("return") > 0), 1).otherwise(0)).alias("correct_long"),
    spark_sum(when((col("prediction") == 2.0) & (col("return") < 0), 1).otherwise(0)).alias("false_long"),

    # Short success/failure
    spark_sum(when((col("prediction") == 0.0) & (col("return") < 0), 1).otherwise(0)).alias("correct_short"),
    spark_sum(when((col("prediction") == 0.0) & (col("return") > 0), 1).otherwise(0)).alias("false_short")
)

stats.toPandas()
```

```
        n  mean_return  std_return  min_return  max_return  total_return  long_counts  neutral_counts  short_counts  correct_lo
0  409396      0.00009    0.058472   -0.962185     35.4625     36.941146        76492          329530          3374         374
```

Train Results

number of articles = 2M
n (number of trades) = 409k
mean (average return) = 9e-5
stdev of return = 0.058
min return = -0.96
max max = 35.46
sum(return) = 36.94

Long correct= 37471 / (37471+36003) = 51% Short correction = 1657 / (1657+1670) = 49.8%

```
# Evaluation on Test set

model = PipelineModel.load("/user/****/sentiment_model_big")
```

```
test_df_with_prediction = model.transform(test_df_filtered)

test_joined_df = test_df_with_prediction.join(
    price_df,
    on=["Date", "Stock_symbol"],
    how="left"
)

test_df_with_strategy_return = test_joined_df.filter(col("return").isNotNull())\
                                         .withColumn(
                                                "strategy_return",
                                                when(col("prediction") == 2.0, col("return"))
                                                .when(col("prediction") == 0.0, -col("return"))
                                                .otherwise(0.0))

stats = test_df_with_strategy_return.select(
    count("strategy_return").alias("n"),
    mean("strategy_return").alias("mean_return"),
    stddev("strategy_return").alias("std_return"),
    min("strategy_return").alias("min_return"),
    max("strategy_return").alias("max_return"),
    spark_sum("strategy_return").alias("total_return"),

    # Prediction counts
    spark_sum(when(col("prediction") == 2.0, 1).otherwise(0)).alias("long_counts"),
    spark_sum(when(col("prediction") == 1.0, 1).otherwise(0)).alias("neutral_counts"),
    spark_sum(when(col("prediction") == 0.0, 1).otherwise(0)).alias("short_counts"),

    # Long success/failure
    spark_sum(when((col("prediction") == 2.0) & (col("return") > 0), 1).otherwise(0)).alias("correct_long"),
    spark_sum(when((col("prediction") == 2.0) & (col("return") < 0), 1).otherwise(0)).alias("false_long"),

    # Short success/failure
    spark_sum(when((col("prediction") == 0.0) & (col("return") < 0), 1).otherwise(0)).alias("correct_short"),
    spark_sum(when((col("prediction") == 0.0) & (col("return") > 0), 1).otherwise(0)).alias("false_short")
)

stats.toPandas()
```

	n	mean_return	std_return	min_return	max_return	total_return	long_counts	neutral_counts	short_counts	correct_lo
0	408404	-0.000052	0.018741	-1.348358	4.612245	-21.287422	76834	328131	3439	376

Test Results

number of articles = 498k
n (number of trades) = 408k
mean (average return) = -5.2e-5
stdev of return = 0.018
min return = -1.34
max max = 4.61
sum(return) = -21.28

Long correct= 37636 / (37636+36281) = 50.9% Short correction = 1676 / (1676+1705) = 49.6%

Long and short correction rates are similar to those of training. This means our model isn't overfitting. The actual returns are less conclusive. Both the test and train results have near 0 average return and 0 Sharpe ratio (average/stdev).

Conclusion

This project aims to utilize the big data tools that we learned to build a custom machine learning pipeline in sentiment driven trading. Based on the results, it appears that extracting sentiments from news is not the most reliable approach. We are not sure sentiments themselves have sufficient values in making trading decisions. However, it was still a valuable project that proves to be capable of handling massive data in a practical example.

```
Start coding or generate with AI.
```