Financial Sentiment Analysis

Problem Statement

Sentiment analysis plays a crucial role in financial markets. Understanding the sentiment of financial news, reports, and discussions can help investors make better decisions. This project aims to classify financial texts into Positive, Negative, or Neutral sentiments using Natural Language Processing (NLP) techniques.

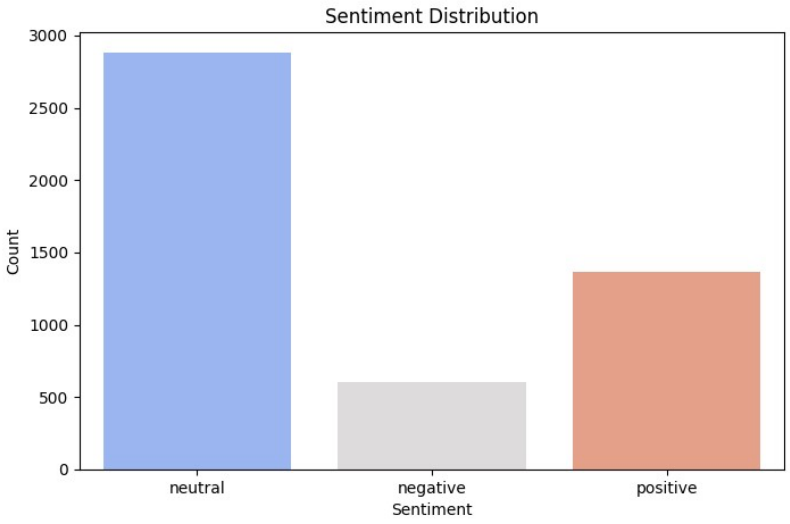
Tech Stack

- Programming Language: Python
- Libraries: NLTK, Pandas, Scikit-learn, Matplotlib, Seaborn, WordCloud
- Machine Learning Model: Logistic Regression
- Dataset: Financial PhraseBank
- Tools: Jupyter Notebook / VS Code

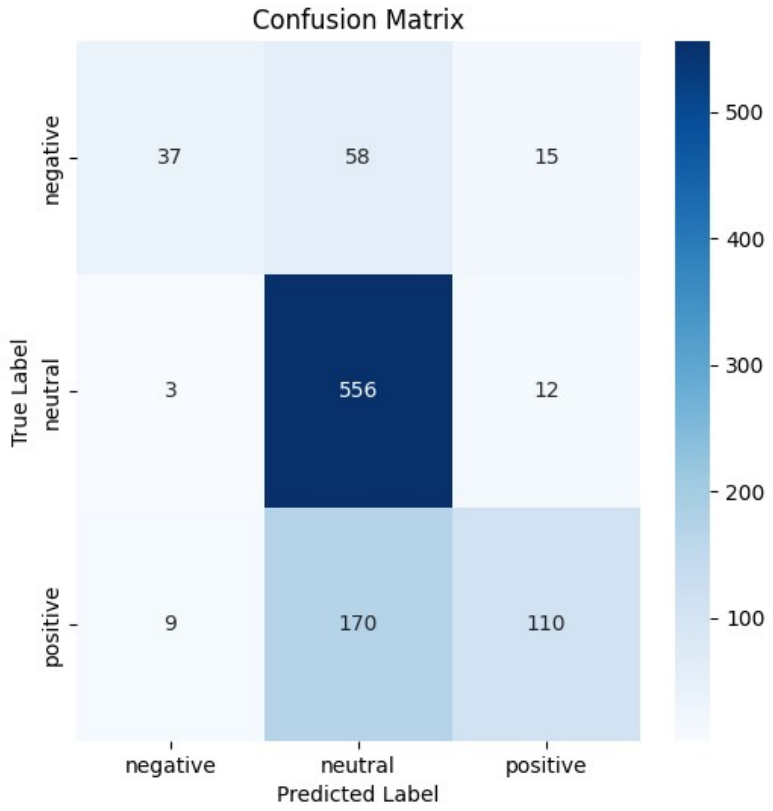
Implementation Steps

1. Data Loading: Load the Financial PhraseBank dataset.
2. Data Preprocessing: Tokenization, stopword removal, and lemmatization.
3. Feature Extraction: Convert text into numerical features using TF-IDF.
4. Model Training: Train a Logistic Regression classifier.
5. Evaluation: Measure accuracy, confusion matrix, and sentiment distribution.

Visualizations



Financial Sentiment Analysis



Ways to Improve the Project

- Improve model performance by using advanced deep learning models like LSTMs or Transformers.
- Expand dataset by including financial news from different sources.
- Implement real-time sentiment analysis by integrating it with web scraping APIs.
- Deploy the model as a web application using Django or Flask.**Project Code**

```
import os import nltk import pandas as pd
import ssl import string import seaborn as
sns import matplotlib.pyplot as plt from
wordcloud import WordCloud from
collections import Counter

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize from nltk.stem import WordNetLemmatizer from
sklearn.feature_extraction.text import TfidfVectorizer from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix from sklearn.preprocessing import LabelEncoder
```

Financial Sentiment Analysis

```
# Fix SSL certificate issue (optional) try:     _create_unverified_https_context =
ssl._create_unverified_context     ssl._create_default_https_context =
_create_unverified_https_context except AttributeError:     pass

# Download necessary NLTK resources nltk.download('punkt')
nltk.download('stopwords') nltk.download('wordnet')

# Ensure dataset exists file_path =
"FinancialPhraseBank.csv" if not
os.path.exists(file_path):
    raise FileNotFoundError(f"The file {file_path} was not found. Please provide the correct path.")

# Load dataset column_names = ["sentiment", "sentence"] df = pd.read_csv(file_path,
names=column_names, encoding="ISO-8859-1")

# Text preprocessing function def preprocess_text(text):     text = str(text).lower()
tokens = word_tokenize(text)     tokens = [word for word in tokens if word.isalnum()]
stop_words = set(stopwords.words("english"))     tokens = [word for word in tokens if
word not in stop_words]     lemmatizer = WordNetLemmatizer()     tokens =
[lemmatizer.lemmatize(word) for word in tokens]     return " ".join(tokens)

# Apply preprocessing df["processed_text"] =
df["sentence"].apply(preprocess_text)

# Convert sentiment labels to numerical format label_encoder = LabelEncoder()
df["sentiment_encoded"] = label_encoder.fit_transform(df["sentiment"])

# TF-IDF Vectorization vectorizer =
TfidfVectorizer(max_features=5000) X =
vectorizer.fit_transform(df["processed_text"]) y =
df["sentiment_encoded"]

# Split data X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)

# Predict and evaluate y_pred = model.predict(X_test) print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Function to predict sentiment def
predict_sentiment(text):
    text = preprocess_text(text)     text_vector = vectorizer.transform([text])
prediction = model.predict(text_vector)     return
label_encoder.inverse_transform(prediction)[0]

# Example prediction new_text = "The company's revenue grew significantly this quarter."
print("Predicted Sentiment:", predict_sentiment(new_text))
```

Financial Sentiment Analysis

```
# --- Exploratory Data Analysis (EDA) --# Ensure 'charts'
directory exists os.makedirs("charts", exist_ok=True)

# Sentiment distribution plt.figure(figsize=(8,5)) sns.countplot(x=df["sentiment"],
palette="coolwarm") plt.title("Sentiment Distribution") plt.xlabel("Sentiment")
plt.ylabel("Count") plt.savefig("charts/sentiment_distribution.png")  # Save chart
plt.close()

# Word Cloud Visualization def
generate_wordcloud(sentiment):
    text = " ".join(df[df["sentiment"] == sentiment]["processed_text"])     wordcloud = WordCloud(width=800, height=400,
background_color="white").generate(text)     plt.figure(figsize=(10, 5))     plt.imshow(wordcloud, interpolation="bilinear")
plt.title(f"Word Cloud for {sentiment} Sentiment")     plt.axis("off")     plt.savefig(f"charts/wordcloud_{sentiment}.png")  #
Save chart     plt.close()

generate_wordcloud("positive") generate_wordcloud("neutral")
generate_wordcloud("negative")

# Confusion Matrix cm = confusion_matrix(y_test, y_pred) plt.figure(figsize=(6,6)) sns.heatmap(cm, annot=True, fmt="d",
cmap="Blues", xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_) plt.xlabel("Predicted Label") plt.ylabel("True
Label") plt.title("Confusion Matrix") plt.savefig("charts/confusion_matrix.png")  # Save chart plt.close()

# Feature Importance Analysis feature_names = vectorizer.get_feature_names_out() importance =
abs(model.coef_).argsort()[0][-10:] print("Most important words for classification:", [feature_names[i] for i in importance])

# Top Words per Sentiment def
get_top_words(sentiment, n=10):
    text = " ".join(df[df["sentiment"] == sentiment]["processed_text"])     words = text.split()
common_words = Counter(words).most_common(n)     return common_words

print("Top words in Positive Sentiment:", get_top_words("positive")) print("Top words in Neutral
Sentiment:", get_top_words("neutral")) print("Top words in Negative Sentiment:",
get_top_words("negative"))
```