```
from google.colab import files
uploaded = files.upload()
```

Choose Files data.csv

- **data.csv**(text/csv) - 745117 bytes, last modified: 6/27/2025 - 100% done

```
Saving data.csv to data.csv
```

```
import pandas as pd

# Load CSV after upload
df = pd.read_csv('data.csv')  # Just the filename
```

```
import pandas as pd
import nltk
import re
from nltk.corpus import stopwords

nltk.download('stopwords')

# Loading dataset
df = pd.read_csv('data.csv')
df.columns = ['text', 'label']

# Preprocessing
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    tokens = re.findall(r'\b[a-z]+\b', text)
    stop_words = set(stopwords.words('english'))
    return [word for word in tokens if word not in stop_words]

df['tokens'] = df['text'].apply(preprocess)
print(df[['text', 'tokens']].head())
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
                                                text  \
0  The GeoSolutions technology will leverage Bene...
1  $ESI on lows, down $1.50 to $2.50 BK a real po...
2  For the last quarter of 2010 , Componenta 's n...
3  According to the Finnish-Russian Chamber of Co...
4  The Swedish buyout firm has sold its remaining...

                                              tokens
0  [geosolutions, technology, leverage, benefon, ...
1                [esi, lows, bk, real, possibility]
2  [last, quarter, componenta, net, sales, double...
3  [according, finnishrussian, chamber, commerce,...
4  [swedish, buyout, firm, sold, remaining, perce...
```

```
!pip install gensim --quiet

from gensim.models import Word2Vec
```

```
# Prepare data: Gensim expects list of token lists
sentences = df['tokens'].tolist()
# CBOW model
model_cbow = Word2Vec(sentences, vector_size=100, window=5, min_count=2, sg=0)
# Skip-gram model
model_sg = Word2Vec(sentences, vector_size=100, window=5, min_count=2, sg=1)
model_cbow.save("cbow_model.model")
model_sg.save("skipgram_model.model")
# Test embeddings
print("CBOW - Similar to 'market':")
print(model_cbow.wv.most_similar('market'))

print("\nSkip-gram - Similar to 'market':")
print(model_sg.wv.most_similar('market'))
```

```
52941895), ('report', 0.9996194839477539), ('day', 0.99961256980896), ('investment', 0.999607264995575)]

405174255), ('tesco', 0.9136277437210083), ('high', 0.912501871585846), ('meeting', 0.91205233335495)]
```

```
!pip install sentence-transformers --quiet

from sentence_transformers import SentenceTransformer

# Load a pre-trained LLM (MiniLM or FinBERT are good options for finance)
model_llm = SentenceTransformer('all-MiniLM-L6-v2')  # general-purpose, fast

# Generate embeddings for each sentence (from original text, not tokens)
df['llm_embedding'] = df['text'].apply(lambda x: model_llm.encode(x))

# Show one embedding vector
print("Sample LLM embedding vector:")
print(df['llm_embedding'].iloc[0][:10])  # show first 10 dimensions
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 1.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 102.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 74.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 41.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 865.1 kB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 4.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 11.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 8.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 1.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 83.6 MB/s eta 0:00:00
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

modules.json: 100% 349/349 [00:00<00:00, 27.6kB/s]

config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 9.69kB/s]

README.md: 10.5k/? [00:00<00:00, 390kB/s]

sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 4.43kB/s]

config.json: 100% 612/612 [00:00<00:00, 38.5kB/s]

model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 101MB/s]

tokenizer_config.json: 100% 350/350 [00:00<00:00, 12.8kB/s]

vocab.txt: 232k/? [00:00<00:00, 6.03MB/s]

tokenizer.json: 466k/? [00:00<00:00, 15.3MB/s]

special_tokens_map.json: 100% 112/112 [00:00<00:00, 8.92kB/s]

config.json: 100% 190/190 [00:00<00:00, 12.8kB/s]

```
Sample LLM embedding vector:
[-0.00591845 -0.06686293  0.05961023 -0.07951813  0.01429671 -0.04960796
  0.01778209  0.02902466 -0.06253076 -0.02630421]
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

import numpy as np
# Convert embedding list to matrix
X = np.vstack(df['llm_embedding'].values)
y = df['label']
# data spliting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# 1. Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)

print("Logistic Regression Report:")
print(classification_report(y_test, y_pred_lr))
# 2. SVM Classifier
```

```
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

print("SVM Report:")
print(classification_report(y_test, y_pred_svm))
# 3. Neural Net (MLP)
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)

print("MLP (Neural Net) Report:")
print(classification_report(y_test, y_pred_mlp))
```

```
Logistic Regression Report:
              precision    recall  f1-score   support

    negative       0.52      0.35      0.42       175
     neutral       0.76      0.83      0.80       622
    positive       0.72      0.72      0.72       372

    accuracy                           0.72      1169
   macro avg       0.67      0.63      0.64      1169
weighted avg       0.71      0.72      0.72      1169

SVM Report:
              precision    recall  f1-score   support

    negative       0.49      0.36      0.42       175
     neutral       0.76      0.85      0.81       622
    positive       0.76      0.71      0.73       372

    accuracy                           0.73      1169
   macro avg       0.67      0.64      0.65      1169
weighted avg       0.72      0.73      0.72      1169

MLP (Neural Net) Report:
              precision    recall  f1-score   support

    negative       0.30      0.27      0.29       175
     neutral       0.74      0.77      0.75       622
    positive       0.77      0.74      0.75       372

    accuracy                           0.69      1169
   macro avg       0.60      0.60      0.60      1169
weighted avg       0.68      0.69      0.68      1169
```

```
import numpy as np
# Helper: average word vectors for a sentence
def sentence_embedding(tokens, model):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(model.vector_size)
# Create embeddings from CBOW
df['cbow_emb'] = df['tokens'].apply(lambda x: sentence_embedding(x, model_cbow))
# Create embeddings from Skip-gram
df['sg_emb'] = df['tokens'].apply(lambda x: sentence_embedding(x, model_sg))
```

```
# Train & evaluate models on CBOW and Skip-gram embeddings
def train_and_report(X_column, label, model_name):
    X = np.vstack(df[X_column].values)
    y = df[label]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    clf = LogisticRegression(max_iter=1000)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print(f"{model_name} Embedding (LogReg):")
    print(classification_report(y_test, y_pred))
# CBOW
train_and_report('cbow_emb', 'label', 'CBOW')
# Skip-gram
train_and_report('sg_emb', 'label', 'Skip-gram')
```

```
CBOW Embedding (LogReg):
              precision    recall  f1-score   support

    negative       0.75      0.02      0.03       175
     neutral       0.54      0.97      0.70       622
    positive       0.45      0.07      0.12       372

    accuracy                           0.54      1169
   macro avg       0.58      0.35      0.28      1169
weighted avg       0.54      0.54      0.41      1169

Skip-gram Embedding (LogReg):
              precision    recall  f1-score   support

    negative       0.45      0.03      0.05       175
     neutral       0.56      0.95      0.70       622
    positive       0.48      0.13      0.20       372

    accuracy                           0.55      1169
   macro avg       0.50      0.37      0.32      1169
weighted avg       0.52      0.55      0.45      1169
```

```
#Evaluation and Comparison of Embeddings
In this part, I compared how different word embeddings affect the performance of sentiment classification mo

- CBOW: Trained from scratch using my dataset
```

- Skip-gram: Also trained from scratch on the same data
- LLM-based embeddings: Generated using the `all-MiniLM-L6-v2` model from SentenceTransformers

For all three embeddings, I trained and tested three ML models:
- Logistic Regression
- SVM (Support Vector Machine)
- MLP (Multi-Layer Perceptron / simple neural network)

I used common evaluation metrics like **accuracy, precision, recall, and F1-score** to see how each model pe