```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()


# ML Algorithms

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression


# DL Models

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, GRU
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


# Evaluation Metrics Libraries

from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay


df = pd.read_csv('/content/sentiment_data.csv')
df
```

	Unnamed: 0	Comment	Sentiment
0	0	lets forget apple pay required brand new iphon...	1
1	1	nz retailers don’t even contactless credit car...	0
2	2	forever acknowledge channel help lessons ideas...	2
3	3	whenever go place doesn’t take apple pay doesn...	0
4	4	apple pay convenient secure easy use used kore...	2
...	...	...	...
241140	241921	crores paid neerav modi recovered congress lea...	0
241141	241922	dear rss terrorist payal gawar modi killing pl...	0
241142	241923	cover interaction forum left	1
241143	241924	big project came india modi dream project happ...	1
241144	241925	ever listen like gurukul discipline maintained...	2

241145 rows × 3 columns

Clean Dataset:

```
df.dropna(axis=0, inplace=True)
df.drop(columns='Unnamed: 0', axis=1, inplace=True)
df
```

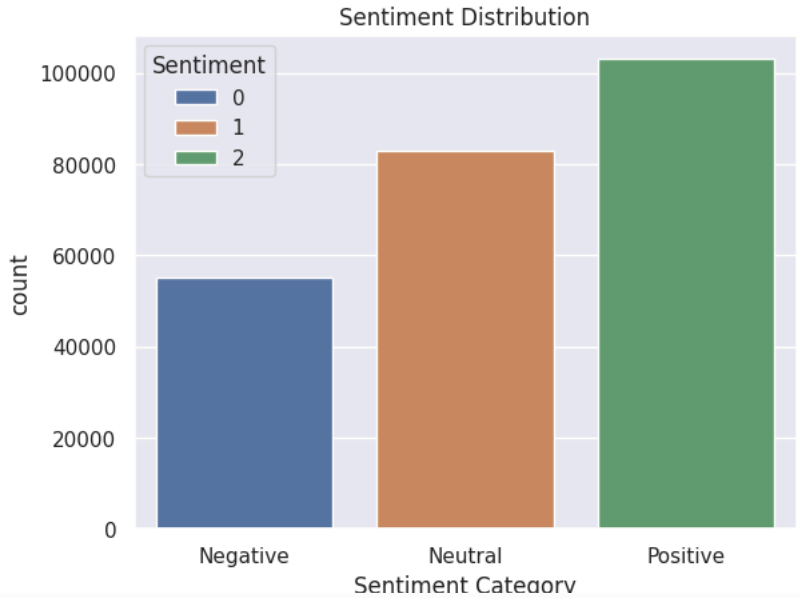
	Comment	Sentiment
0	lets forget apple pay required brand new iphon...	1
1	nz retailers don’t even contactless credit car...	0
2	forever acknowledge channel help lessons ideas...	2
3	whenever go place doesn’t take apple pay doesn...	0
4	apple pay convenient secure easy use used kore...	2
...	...	...
241140	crores paid neerav modi recovered congress lea...	0
241141	dear rss terrorist payal gawar modi killing pl...	0
241142	cover interaction forum left	1
241143	big project came india modi dream project happ...	1
241144	ever listen like gurukul discipline maintained...	2

240928 rows × 2 columns

Data Exploration:

```
sns.countplot(data=df, x= 'Sentiment',hue= 'Sentiment', palette='deep')
plt.xticks(ticks=[0, 1, 2], labels=['Negative', 'Neutral', 'Positive'])
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment Category')
```

```
Text(0.5, 0, 'Sentiment Category')
```




```
# Feature Selection
X = df['Comment']
y = df['Sentiment']
```

```
#Spliting into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

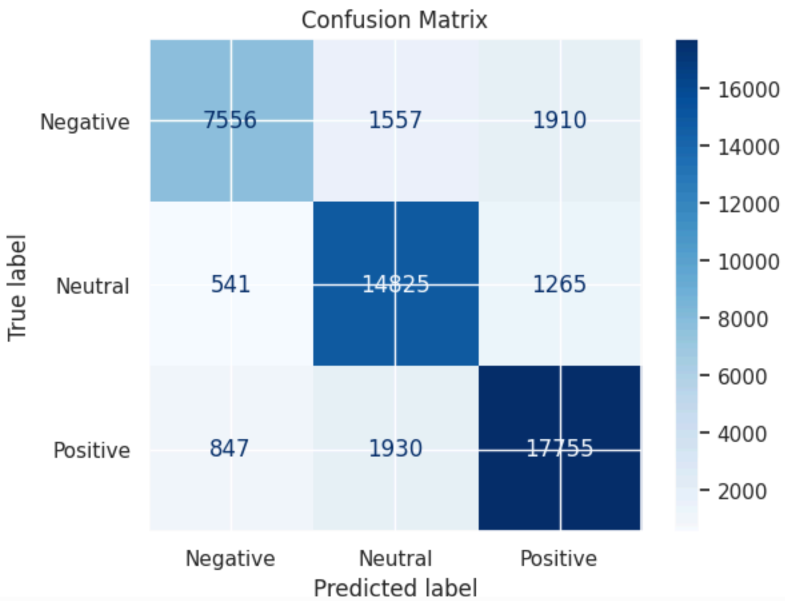
```
# Transforming the text data using TF-IDF:
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```
# Defining a dectionary of classification models
models = {
    "Logistic Regression": LogisticRegression(max_iter=2000),
    "Naive Bayes": MultinomialNB(),
    "KNN": KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(),
}
```

```
for name, model in models.items():
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    print(f'{name} Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Logistic Regression Accuracy: 0.789316398954053
Naive Bayes Accuracy: 0.6694683102975968
KNN Accuracy: 0.46119204748267134
Random Forest Accuracy: 0.8329390279334247
```

```
# confusion matrix for each model to analyze
# how well it distinguishes between negative, neutral, and positive sentiments.
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negative', 'Neutral', 'Positive'])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



DL Models:

1- LSTM

```
# Tokenization
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(df['Comment'])
sequences = tokenizer.texts_to_sequences(df['Comment'])


# Padding
padded = pad_sequences(sequences, maxlen=100)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(padded, y, test_size=0.2, random_state=42, stratify=y)

# Build LSTM Model
lstm = Sequential()
lstm.add(Embedding(input_dim=10000, output_dim=128))
lstm.add(LSTM(46))
lstm.add(Dense(len(set(y)), activation='softmax'))

# Compile & Train
lstm.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
lstm.fit(X_train, y_train, batch_size=128, epochs=5, validation_data=(X_test, y_test))

# Evaluate
loss, accuracy = lstm.evaluate(X_test, y_test)
print(f'LSTM Accuracy: {accuracy:.2f}')
```

```
Epoch 1/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 361s 233ms/step - accuracy: 0.7050 - loss: 0.6977 - val_accuracy: 0.8313 - val_loss: 0.4571
Epoch 2/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 338s 224ms/step - accuracy: 0.8446 - loss: 0.4183 - val_accuracy: 0.8450 - val_loss: 0.4295
Epoch 3/5
```

```
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 409s 242ms/step - accuracy: 0.8713 - loss: 0.3569 - val_accuracy: 0.8468 - val_loss: 0.4296
Epoch 4/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 361s 229ms/step - accuracy: 0.8897 - loss: 0.3049 - val_accuracy: 0.8455 - val_loss: 0.4452
Epoch 5/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 397s 239ms/step - accuracy: 0.9081 - loss: 0.2597 - val_accuracy: 0.8462 - val_loss: 0.4671
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.8488 - loss: 0.4631
LSTM Accuracy: 0.85
```

2- GRU:

```
# Build GRU Model
gru = Sequential()
gru.add(Embedding(input_dim=10000, output_dim=128))
gru.add(GRU(64))
gru.add(Dense(3, activation='softmax'))

# Compile & Train
gru.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
gru.fit(X_train, y_train, batch_size=128, epochs=5, validation_data=(X_test, y_test))

# Evaluate
loss, accuracy = gru.evaluate(X_test, y_test)
print(f'GRU Accuracy: {accuracy:.2f}')
```

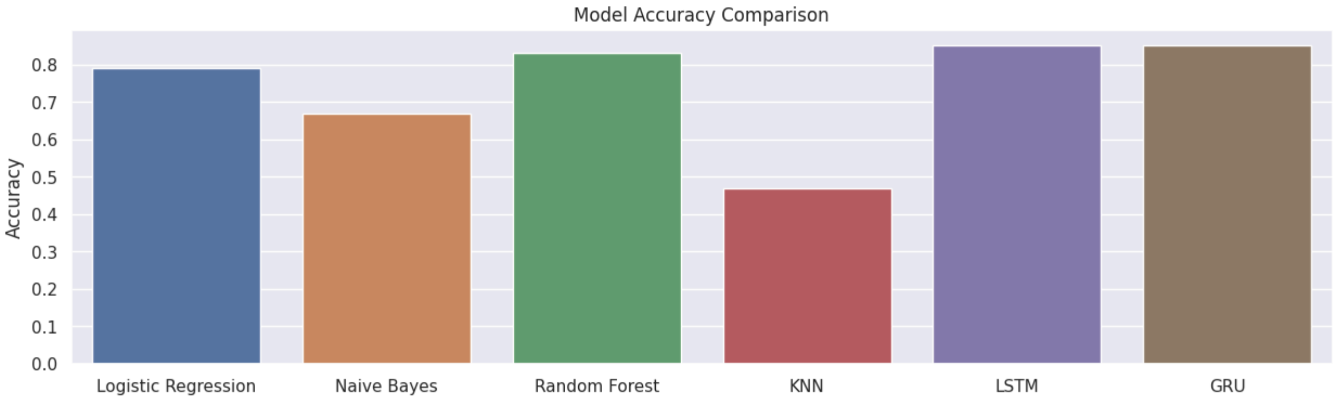
```
Epoch 1/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 523s 341ms/step - accuracy: 0.7146 - loss: 0.6889 - val_accuracy: 0.8314 - val_loss: 0.4621
Epoch 2/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 564s 342ms/step - accuracy: 0.8448 - loss: 0.4210 - val_accuracy: 0.8441 - val_loss: 0.4334
Epoch 3/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 548s 333ms/step - accuracy: 0.8686 - loss: 0.3646 - val_accuracy: 0.8477 - val_loss: 0.4265
Epoch 4/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 516s 343ms/step - accuracy: 0.8880 - loss: 0.3141 - val_accuracy: 0.8486 - val_loss: 0.4342
Epoch 5/5
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 547s 333ms/step - accuracy: 0.9040 - loss: 0.2710 - val_accuracy: 0.8463 - val_loss: 0.4605
1506/1506 ━━━━━━━━━━━━━━━━━━━━ 39s 26ms/step - accuracy: 0.8491 - loss: 0.4558
GRU Accuracy: 0.85
```

```
# Finally: model accuracy comparison
plt.figure(figsize=(15,4))
model_scores = {
    "Logistic Regression": 0.79,
    "Naive Bayes": 0.67,
    "Random Forest": 0.83,
    "KNN": 0.47,
    'LSTM': 0.85,
    'GRU': 0.85
                }
sns.barplot(x= list(model_scores.keys()), y= list(model_scores.values()), palette='deep')
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.show()
```

```
<ipython-input-27-2436618548>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x= list(model_scores.keys()), y= list(model_scores.values()), palette='deep')
```



Summary

This notebook demonstrates a comprehensive sentiment analysis pipeline:

1. *Data Preparation*: Cleaned and explored a dataset of 241,145 comments with 3 sentiment classes (Negative, Neutral, Positive).
2. *Feature Engineering*: Applied TF-IDF vectorization for ML models and tokenization/padding for DL models.
3. *Model Training*: Evaluated 4 ML models (Logistic Regression, Naive Bayes, KNN, Random Forest) and 2 DL models (LSTM, GRU).
4. *Results*:
    - Random Forest achieved 83.3% accuracy, while LSTM/GRU reached 85% accuracy.
    - Confusion matrices and visualizations provided insights into model performance.
5. *Conclusion*: Deep Learning models slightly outperformed traditional ML, with GRU being computationally efficient.