Opinion Mining using Recurrent Neural Networks:

Recurrent Neural Networks (RNNs) excel in sequence tasks such as sentiment analysis due to their ability to capture context from sequential data. In this article we will be apply RNNs to analyze the sentiment of customer reviews from Swiggy food delivery platform. The goal is to classify reviews as positive or negative for providing insights into customer experiences.

We will conduct a Sentiment Analysis using the TensorFlow framework:

Step 1: Importing Libraries and Dataset

Here we will be importing numpy, pandas, Regular Expression (RegEx), scikit learn and tenserflow.

```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Embedding
```

Step 2: Loading Dataset

We will be using swiggy dataset of customer reviews. You can download dataset from here.

```
data = pd.read_csv('swiggy.csv')
print("Columns in the dataset:")
print(data.columns.tolist())
```

Output:

Columns in the dataset:
['ID', 'Area', 'City', 'Restaurant Price', 'Avg Rating', 'Total Rating', 'Food Item', 'Food Type', 'Delivery Time', 'Review']

Step 3: Text Cleaning and Sentiment Labeling

data['sentiment']: Uses Avg Rating to generate binary labels (positive if rating >3.5)

```
data["Review"] = data["Review"].str.lower()
data["Review"] = data["Review"].replace(r'[^a-z0-9\s]', '', regex=True)

data['sentiment'] = data['Avg Rating'].apply(lambda x: 1 if x > 3.5 else 0)
data = data.dropna()
```

Tokenization and Padding

```
max_features = 5000
max_length = 200

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(data["Review"])
X = pad_sequences(tokenizer.texts_to_sequences(data["Review"]), maxlen=max_length)
y = data['sentiment'].values
```

- Tokenizer: Converts words into integer sequences.
- Padding: Ensures all input sequences have the same length (max_length).

Note: These concepts are a not a part of RNN but are done to make model prediction better. You can refer to tokenization and padding for more details.

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.1, random_state=42, stratify=y_train
)
```

Build RNN Model

```
model = Sequential([
    Embedding(input_dim=max_features, output_dim=16, input_length=max_length),
    SimpleRNN(64, activation='tanh', return_sequences=False),
    Dense(1, activation='sigmoid')
])

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

- Embedding Layer: Converts integer sequences into dense vectors (16 dimensions).
- RNN Layer: Processes sequence data with 64 units and tanh activation.
- Output Layer: Predicts sentiment probability using sigmoid activation.

Train & Evaluate Model

```
history = model.fit(    X_train, y_train,
    epochs=5,
    batch_size=32,
    validation_data=(X_val, y_val),
    verbose=1
)

score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test accuracy: {score[1]:.2f}")
```

Output:

Test accuracy: 0.72

- Epochs: Number of training iterations i.e 5
- Batch Size: Processes 32 samples per gradient update.