**Assignment one – Emoji Predictor**

## Step 1: Understanding the Problem

Need to predict emojis based on a dataset, which could contain text, sentiment, or other features. This can be treated as a classification problem, where the model predicts which emoji is best associated with the input data (e.g., a sentence or a word).

## Step 2: Dataset Preparation

You need to have a dataset that has some structure. For example:

- **Text Data** (e.g., tweets, sentences, or reviews).
- **Emoji Labels** (the emoji corresponding to the text).

A simple example might look like this:

| Text | Emoji |
|---|---|
| "I'm so happy!" | 😊 |
| "I feel sad." | 😞 |
| "That was awesome" | 😎 |
| "I'm angry" | 😡 |

Now, we can proceed to build a model that can predict emojis from the text.

## Step 3: Preprocessing the Data

Before building any model, you need to preprocess the text and labels. This involves:

1. **Cleaning the text**: Remove punctuation, lowercasing, and tokenization.
2. **Vectorizing the text**: Converting words into numerical representations (e.g., using TF-IDF or word embeddings).
3. **Encoding the emojis**: Convert the emoji labels into numerical values (e.g., one-hot encoding or label encoding).

## Step 4: Building the Model

For simplicity, we'll use a **machine learning** approach like a **Naive Bayes** classifier or a **deep learning** approach like a **Neural Network** (if the dataset is large enough).

We'll first write code using the **Naive Bayes** classifier with **TF-IDF** vectorization for simplicity.

### Step 5: Code Implementation

### Install Required Libraries

First, necessary libraries installed:

```
pip install pandas scikit-learn
```

### Import Required Libraries

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

### Load and Preprocess the Dataset

```
data = {
    'Text': ["I'm so happy!", "I feel sad.", "That was awesome", "I'm angry", "Feeling great today!", "I'm frustrated","Feeling chill", "This makes me so angry", "This is too much", "I could use a little more sleep", "I'm really shocked", "Today is awesome!","This is so frustrating!", "I'm so at peace", "What a disaster!", "Feeling fantastic", "That was so cool!", "I'm feeling very emotional", "I want to cry", "So many good things are happening", "I can't stop smiling!"],

    'Emoji': ['😊', '😞', '😎', '😡', '😁', '😭', '😳', '😵', '💧', '😇', '💀', '😫', '👀', '😱', '😠', '😔', '😅', '🤣', '💀', '😞', '😇',]
}
```

```
df = pd.DataFrame(data)

# Optional: Clean the text (convert to lowercase, remove punctuation, etc.)
df['Text'] = df['Text'].str.lower()

# Encode the emojis (labels) as numeric values
label_encoder = LabelEncoder()
df['Emoji'] = label_encoder.fit_transform(df['Emoji'])

# Split the data into features (X) and labels (y)
X = df['Text']
y = df['Emoji']
```

### Text Vectorization (TF-IDF)

We'll use the **TF-IDF** vectorizer to convert the text into numerical features.

```
# Convert text into TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english')
X_vectorized = vectorizer.fit_transform(X)
```

## 5. Train-Test Split

We'll split the data into training and testing sets.

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y, test_size=0.2, random_state=42)
```

## 6. Train a Model (Naive Bayes)

Now, let's train a **Naive Bayes** classifier.

```
# Train a Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)
```

## 7. Evaluate the Model

After training, we'll evaluate the model using the test data.

```
# Make predictions on the test data
y_pred = model.predict(X_test)
# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Decode predicted emoji labels
predicted_emojis = label_encoder.inverse_transform(y_pred)
print("Predicted Emojis:", predicted_emojis)
```

## 8. Predicting New Text

Now, you can use the trained model to predict emojis for new text.

```
# Function to predict emoji for new text
def predict_emoji(text):
  text = text.lower()  # Clean the text (lowercase)
  vectorized_text = vectorizer.transform([text])
 prediction = model.predict(vectorized_text)
```

```
    emoji = label_encoder.inverse_transform(prediction)
    return emoji[0]
# Test the function
new_text = "I feel amazing!"
predicted_emoji = predict_emoji(new_text)
print(f"The predicted emoji for '{new_text}' is {predicted_emoji}")
```

## Output

```
chill
The predicted emoji for 'chill' is 🌼
```

```
Lovely
The predicted emoji for 'Lovely' is 💙
```

```
Angry
The predicted emoji for 'Angry' is 👿
```

## Step 6: Conclusion

This code should help you build a basic machine learning model that predicts emojis based on text. You can extend it by:

- Adding more data to improve accuracy.
- Experimenting with other models (e.g., Logistic Regression, SVM, or Neural Networks).
- Tuning hyperparameters for better performance.