Problem Statement:
This project focuses on developing an English-to-French translation system using Recurrent Neural Networks (RNN). The model will learn language patterns and translate English sentences into their French equivalents. A dataset of English-French sentence pairs will be used for training and evaluation. The system aims to demonstrate the effectiveness of RNNs in handling sequence-to-sequence tasks like machine translation. The final solution will provide accurate translations for simple sentences.

Dataset:

Dataset contains 2 columns:

1. English Word
2. Corresponding French Word

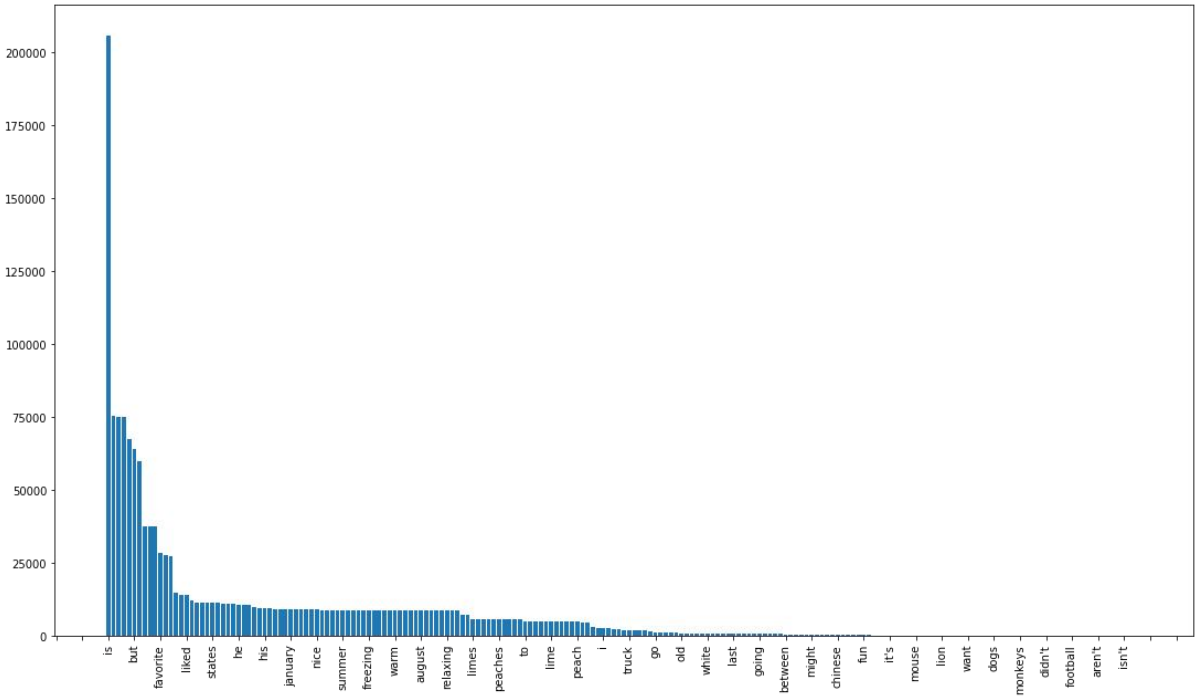
Implementation and Code:

```
import nltk

nltk.download('punkt')

nltk.download("stopwords")

from collections import Counter

import operator

import plotly.express as px

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from wordcloud import WordCloud, STOPWORDS

import nltk

import re

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize, sent_tokenize

import gensim

from gensim.utils import simple_preprocess

from gensim.parsing.preprocessing import STOPWORDS

from tensorflow.keras.preprocessing.text import one_hot, Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, TimeDistributed, RepeatVector, Embedding, Input, LSTM, Conv1D, MaxPool1D, Bidirectional

from tensorflow.keras.models import Model
```

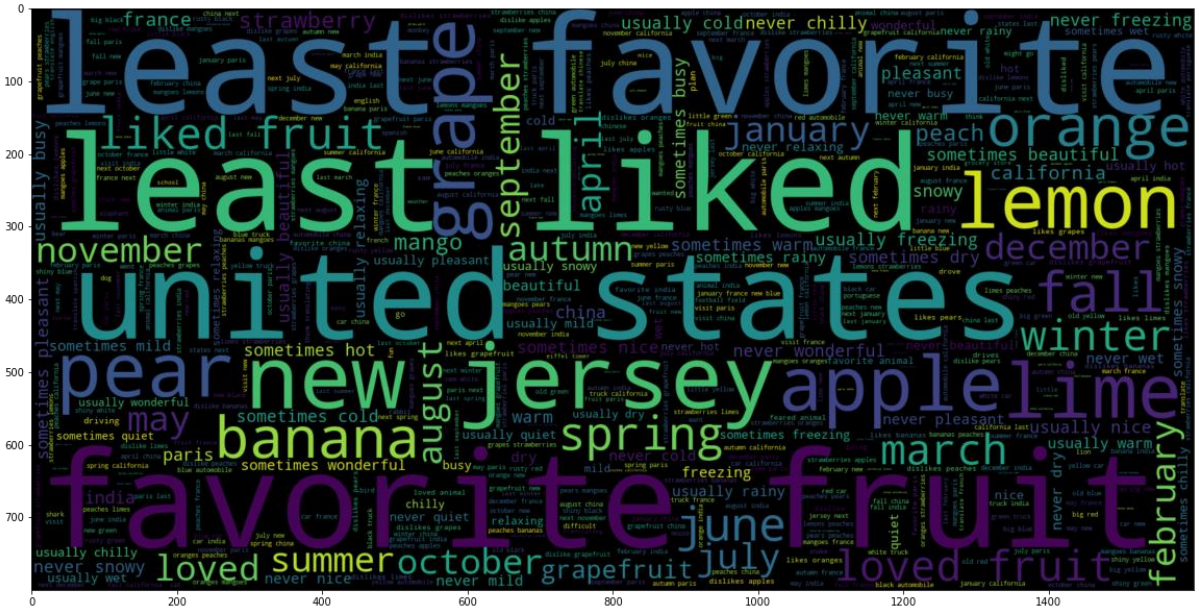
```
df_english = pd.read_csv('small_vocab_en.csv', sep = '/t', names = ['english'], engine = "python")
df_french = pd.read_csv('small_vocab_fr.csv', sep = '/t', names = ['french'],  engine = "python")
df_english.info()
df_french.info()
df = pd.concat([df_english, df_french], axis = 1)
def remove_punc(x):
  return re.sub('[!#?,.:";]', '', x)
In [8]:
df['french'] = df['french'].apply(remove_punc)
df['english'] = df['english'].apply(remove_punc)
In [9]:
english_words = []
french_words  = []
In [10]:
# function to get the list of unique words

def get_label_superset(x, word_list):
   for label in x.split():
      if label not in word_list:
         word_list.append(label)
In [11]:
df['english'].apply(lambda x: get_label_superset(x, english_words))
df['french'].apply(lambda x: get_label_superset(x, french_words))

# number of unique words in english

total_english_words  = len(english_words)
total_english_words
# number of unique words in french

total_french_words = len(french_words)
total_french_words
import matplotlib.ticker as ticker
In [21]:
```

```
fig = plt.figure()
ax  = plt.gca()

ax.bar(x=english_words, height=english_counts)
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
plt.xticks(rotation=90)

fig.set_size_inches(18.5, 10.5)
```

output:




```
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000, width = 1600, height = 800 ).generate(" ".join(df.english))
plt.imshow(wc, interpolation = 'bilinear')
```



 # Sequential Model

model = Sequential()

 # embedding layer

model.add(Embedding(english_vocab_size, 256, input_length = maxlen_english, mask_zero = **True**))

 # encoder

model.add(LSTM(256))

 # decoder

 # repeatvector repeats the input for the desired number of times to change

 # 2D-array to 3D array. For example: (1,256) to (1,23,256)

model.add(RepeatVector(maxlen_french))

model.add(LSTM(256, return_sequences= **True**))

model.add(TimeDistributed(Dense(french_vocab_size, activation ='softmax')))

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

model.summary()

 # change the shape of target from 2D to 3D

f

y_train = np.expand_dims(y_train, axis = 2)

y_train.shape

```
# train the model
model.fit(x_train, y_train, batch_size=1024, validation_split= 0.1, epochs=10)
# save the model
model.save("weights.h5")
# function to make prediction


def prediction(x, x_tokenizer = x_tokenizer, y_tokenizer = y_tokenizer):
    predictions = model.predict(x)[0]
    id_to_word = {id: word for word, id in y_tokenizer.word_index.items()}
    id_to_word[0] = ''
    return ' '.join([id_to_word[j] for j in np.argmax(predictions,1)])
def pad_to_text(padded, tokenizer):
    id_to_word = {id: word for word, id in tokenizer.word_index.items()}
    id_to_word[0] = ''
    return ' '.join([id_to_word[j] for j in padded])
for i in range(5):
  print('Original English word - {}\n'.format(pad_to_text(x_test[i], x_tokenizer)))
  print('Original French word - {}\n'.format(pad_to_text(y_test[i], y_tokenizer)))
  print('Predicted French word - {}\n\n\n\n'.format(prediction(x_test[i:i+1])))
```

OUTPUT:

Original English word - our favorite fruit is the lime but my favorite is the orange

Original French word - notre fruit préféré est la chaux mais mon préféré est l'orange

Predicted French word - son fruit préféré est la fraise mais son préféré est l'orange

Original English word - she plans to visit paris in may

Original French word - elle envisage de visiter paris en mai

Predicted French word - il envisage de en en en en mai

Result:

Thus the English To French language Translation using RNN and LSTM is implemented successfully.