```
## Question 1)
```

```
import pandas as pd
import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount= True)
```

```
     Mounted at /content/drive
```

```
df = pd.read_csv('/content/drive/MyDrive/Natural Language Processing Datasets/spam.csv', encoding='latin-1')
```

```
df.rename(columns={'v1': 'label', 'v2':'message'}, inplace=True)
```

```
df['label'].value_counts()/df['label'].count()*100
```

```
     ham     86.593683
     spam    13.406317
     Name: label, dtype: float64
```

```
## Question 2)The best metric to evaluate the model is F1 score. Since it is a highly  imbalance data we can achieve a 87% accuracy by
## Question 3) Pipeline 1 sparse embeding
```

```
df.count()
dfsub = df.sample(n = 500)
```

```
import spacy
import nltk
nltk.download('stopwords')  # Download the stopwords corpus
from nltk.corpus import stopwords as nltk_stopwords  # Stopwords corpus
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!
```

```
Tfidf = TfidfVectorizer( min_df= 1, max_df= 65)
```

```
tfidf_vector = Tfidf.fit_transform(dfsub['message'])
```

```
tfidf_vector.shape
```

```
     (500, 2100)
```

```
tfidf_df = pd.DataFrame(tfidf_vector.toarray(), columns = Tfidf.get_feature_names_out())
```

```
tfidf_df.round(4)
```

	00	000	02	03	04	0578	06	07815296484	07821230901	07xxxxxxxxx	...	yup
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
495	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
496	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
497	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
498	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
499	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

500 rows × 2100 columns

```
dfsub['label'].value_counts()/dfsub['label'].count()*100
```

```
ham     84.2
spam    15.8
Name: label, dtype: float64
```

```
dfsub = dfsub.reset_index()
tfidf_df['label'] = dfsub['label']
tfidf_df.head()
```

	00	000	02	03	04	0578	06	07815296484	07821230901	07xxxxxxxxx	...	zaher
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

5 rows × 2101 columns

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

```
LR = LogisticRegression(class_weight={'ham': 1.3, 'spam': 8.6})
```

```
from sklearn.model_selection import train_test_split
```

```
X = tfidf_df.drop(['label'], axis = 1)
y = tfidf_df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
LR.fit(X_train, y_train)
```

```
▾                      LogisticRegression
LogisticRegression(class_weight={'ham': 1.3, 'spam': 8.6})
```

```
predictions = LR.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, classification_report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99        81
        spam       1.00      0.89      0.94        19

    accuracy                           0.98       100
   macro avg       0.99      0.95      0.97       100
weighted avg       0.98      0.98      0.98       100
```

```
## Second Pipeline

## Feature Extraction and preprocessing
```

```
dfsub2 = df.sample( n =500 , random_state= 43)
dfsub2.head(5)
```

	label	message	Unnamed: 2	Unnamed: 3	Unnamed: 4
2347	ham	But i dint slept in afternoon.	NaN	NaN	NaN
676	ham	Maybe?! Say hi to and find out if got his ca...	NaN	NaN	NaN
143	ham	I know you are. Can you pls open the back?	NaN	NaN	NaN
1077	ham	Yep, by the pretty sculpture	NaN	NaN	NaN

```
from sklearn.base import BaseEstimator, TransformerMixin
from bs4 import BeautifulSoup
import re
import spacy
import numpy as np
from nltk.stem.porter import PorterStemmer
import os

class SpacyPreprocessor(BaseEstimator, TransformerMixin):

    def __init__(self, model, *, batch_size = 64, lemmatize=True, lower=True, remove_stop=True,
                remove_punct=True, remove_email=False, remove_url=False, remove_num=False, stemming = False,
                add_user_mention_prefix=True, remove_hashtag_prefix=False, basic_clean_only=False):

        self.model = model
        self.batch_size = batch_size
        self.remove_stop = remove_stop
        self.remove_punct = remove_punct
        self.remove_num = remove_num
        self.lower = lower
        self.add_user_mention_prefix = add_user_mention_prefix
        self.remove_hashtag_prefix = remove_hashtag_prefix
        self.basic_clean_only = basic_clean_only

        if lemmatize and stemming:
            raise ValueError("Only one of 'lemmatize' and 'stemming' can be True.")

        # Validate basic_clean_only option
        if self.basic_clean_only and (lemmatize or lower or remove_stop or remove_punct or remove_num or stemming or
                                      add_user_mention_prefix or remove_hashtag_prefix):
            raise ValueError("If 'basic_clean_only' is set to True, other processing options must be set to False.")

        # Assign lemmatize and stemming

        self.lemmatize = lemmatize
        self.stemming = stemming

    def basic_clean(self, text):
        soup = BeautifulSoup(text, "html.parser")
        text = soup.get_text()
        text = re.sub(r'[\n\r]', ' ', text)
        return text.strip()

    def spacy_preprocessor(self, texts):
        final_result = []
        nlp = spacy.load(self.model)

        # Disable unnecessary pipelines in spaCy model
        if self.lemmatize:
            # Disable parser and named entity recognition
            disabled_pipes = ['parser', 'ner']
        else:
            # Disable tagger, parser, attribute ruler, lemmatizer and named entity recognition
            disabled_pipes = ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']

        with nlp.select_pipes(disable=disabled_pipes):
          # Modify tokenizer behavior based on user_mention_prefix and hashtag_prefix settings
          if self.add_user_mention_prefix or self.remove_hashtag_prefix:
              prefixes = list(nlp.Defaults.prefixes)
              if self.add_user_mention_prefix:
                  prefixes += ['@']  # Treat '@' as a separate token
              if self.remove_hashtag_prefix:
                  prefixes.remove(r'#')  # Don't separate '#' from the following text
              prefix_regex = spacy.util.compile_prefix_regex(prefixes)
              nlp.tokenizer.prefix_search = prefix_regex.search

          # Process text data in parallel using spaCy's nlp.pipe()
          for doc in nlp.pipe(texts, batch_size=self.batch_size):
              filtered_tokens = []
              for token in doc:
                  # Check if token should be removed based on specified filters
                  if self.remove_stop and token.is_stop:
                      continue
                  if self.remove_punct and token.is_punct:
                      continue
                  if self.remove_num and token.like_num:
                      continue

                  # Append the token's text, lemma, or stemmed form to the filtered_tokens list
                  if self.lemmatize:
                      filtered_tokens.append(token.lemma_)
                  elif self.stemming:
                      filtered_tokens.append(PorterStemmer().stem(token.text))
                  else:
                      filtered_tokens.append(token.text)

              # Join the tokens and apply lowercasing if specified
              text = ' '.join(filtered_tokens)
              if self.lower:
                  text = text.lower()
              final_result.append(text.strip())

        return final_result


    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        try:
            if not isinstance(X, (list, np.ndarray)):
                raise TypeError(f'Expected list or numpy array, got {type(X)}')

            x_clean = [self.basic_clean(text).encode('utf-8', 'ignore').decode() for text in X]

            # Check if only basic cleaning is required
            if self.basic_clean_only:
                return x_clean  # Return the list of basic-cleaned texts

            x_clean_final = self.spacy_preprocessor(x_clean)
            return x_clean_final

        except Exception as error:
            print(f'An exception occurred: {repr(error)}')


preprocessor = SpacyPreprocessor(model='en_core_web_sm', batch_size=64, lemmatize=False, lower=True,
                                    remove_stop=True, remove_punct=True, remove_num=False, stemming=False,
                                    add_user_mention_prefix=True, remove_hashtag_prefix=True, basic_clean_only=False)


cleaned_text = preprocessor.fit_transform(dfsub2['message'].values)
```

```
<ipython-input-97-41e9fca79935>:39: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to
  soup = BeautifulSoup(text, "html.parser")
```

```python
from sklearn.base import TransformerMixin, BaseEstimator
import numpy as np
import spacy
import re
import sys
import os
from pathlib import Path


class ManualFeatures(TransformerMixin, BaseEstimator):


    def __init__(self, spacy_model, batch_size = 64, pos_features = True, ner_features = True, text_descriptive_features = True):

        self.spacy_model = spacy_model
        self.batch_size = batch_size
        self.pos_features = pos_features
        self.ner_features = ner_features
        self.text_descriptive_features = text_descriptive_features

    def get_cores(self):
        """
        Get the number of CPU cores to use in parallel processing.
        """
        # Get the number of CPU cores available on the system.
        num_cores = os.cpu_count()
        if num_cores < 3:
            use_cores = 1
        else:
            use_cores = num_cores // 2 + 1
        return num_cores

    def get_pos_features(self, cleaned_text):

        nlp = spacy.load(self.spacy_model)
        noun_count = []
        aux_count = []
        verb_count = []
        adj_count =[]

        # Disable the lemmatizer and NER pipelines for improved performance
        disabled_pipes = ['lemmatizer', 'ner']
        with nlp.select_pipes(disable=disabled_pipes):
            n_process = self.get_cores()
            for doc in nlp.pipe(cleaned_text, batch_size=self.batch_size, n_process=n_process):
                # Extract nouns, auxiliaries, verbs, and adjectives from the document
                nouns = [token.text for token in doc if token.pos_ in ["NOUN","PROPN"]]
                auxs =  [token.text for token in doc if token.pos_ in ["AUX"]]
                verbs =  [token.text for token in doc if token.pos_ in ["VERB"]]
                adjectives =  [token.text for token in doc if token.pos_ in ["ADJ"]]

                # Store the count of each type of word in separate lists
                noun_count.append(len(nouns))
                aux_count.append(len(auxs))
                verb_count.append(len(verbs))
                adj_count.append(len(adjectives))

        # Stack the count lists vertically to form a 2D numpy array
        return np.transpose(np.vstack((noun_count, aux_count, verb_count, adj_count)))



    def get_ner_features(self, cleaned_text):
        nlp = spacy.load(self.spacy_model)
        count_ner = []

        # Disable the tok2vec, tagger, parser, attribute ruler, and lemmatizer pipelines for improved performance
        disabled_pipes = ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer']
        with nlp.select_pipes(disable=disabled_pipes):
            n_process = self.get_cores()
            for doc in nlp.pipe(cleaned_text, batch_size=self.batch_size, n_process=n_process):
                ners = [ent.label_ for ent in doc.ents]
                count_ner.append(len(ners))

        # Convert the list of NER counts to a 2D numpy array
        return np.array(count_ner).reshape(-1, 1)


    def get_text_descriptive_features(self, cleaned_text):
        list_count_words = []
        list_count_characters = []
        list_count_characters_no_space = []
        list_avg_word_length = []
        list_count_digits = []
        list_count_numbers = []
        list_count_sentences = []

        nlp = spacy.load(self.spacy_model)
        disabled_pipes = ['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
        with nlp.select_pipes(disable=disabled_pipes):
            if not nlp.has_pipe('sentencizer'):
                nlp.add_pipe('sentencizer')
            n_process = self.get_cores()
            for doc in nlp.pipe(cleaned_text, batch_size=self.batch_size, n_process=n_process):
                count_word = len([token for token in doc if not token.is_punct])
                count_char = len(doc.text)
                count_char_no_space = len(doc.text_with_ws.replace(' ', ''))
                avg_word_length = count_char_no_space / (count_word + 1)
                count_numbers = len([token for token in doc if token.is_digit])
                count_sentences = len(list(doc.sents))

                list_count_words.append(count_word)
                list_count_characters.append(count_char)
                list_count_characters_no_space.append(count_char_no_space)
                list_avg_word_length.append(avg_word_length)
                list_count_numbers.append(count_numbers)
                list_count_sentences.append(count_sentences)

        text_descriptive_features = np.vstack((list_count_words, list_count_characters, list_count_characters_no_space, list_avg_word_le
                                            list_count_numbers, list_count_sentences))
        return np.transpose(text_descriptive_features)


    def fit(self, X, y=None):

        return self


    def transform(self, X, y=None):

        try:
            # Check if the input data is a list or numpy array
            if not isinstance(X, (list, np.ndarray)):
                raise TypeError(f"Expected list or numpy array, got {type(X)}")


            feature_names = []

            if self.text_descriptive_features:
                text_descriptive_features = self.get_text_descriptive_features(X)
                feature_names.extend(['count_words', 'count_characters',
                                      'count_characters_no_space', 'avg_word_length',
                                      'count_numbers', 'count_sentences'])
            else:
                text_descriptive_features = np.empty(shape=(0, 0))

            if self.pos_features:
                pos_features = self.get_pos_features(X)
                feature_names.extend(['noun_count', 'aux_count', 'verb_count', 'adj_count'])
            else:
                pos_features = np.empty(shape=(0, 0))

            if self.ner_features:
                ner_features = self.get_ner_features(X)
                feature_names.extend(['ner'])
            else:
                ner_features = np.empty(shape=(0, 0))

            # Stack the feature arrays horizontally to form a single 2D numpy array
            return np.hstack((text_descriptive_features, pos_features,  ner_features)), feature_names

        except Exception as error:
            print(f'An exception occured: {repr(error)}')

featurizer = ManualFeatures(spacy_model='en_core_web_sm', batch_size =3)


X_train_features, feature_names = featurizer.fit_transform(cleaned_text )


fext_df = pd.DataFrame(X_train_features, columns=feature_names )
```

```
dfsub2 = dfsub2.reset_index()
dffsub2 = dfsub2.drop(['index'], axis = 1)
dfsub.head()
fext_df['label'] = dfsub2['label']
```

```
fext_df.head(5)
```

	count_words	count_characters	count_characters_no_space	avg_word_length	count_r
0	3.0	20.0	18.0	4.500000	
1	10.0	52.0	41.0	3.727273	
2	3.0	13.0	11.0	2.750000	
3	3.0	20.0	18.0	4.500000	
4	2.0	8.0	7.0	2.333333	

```
X = fext_df.drop(['label'], axis = 1)
y = fext_df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
LR = LogisticRegression(class_weight={'ham': 1.3, 'spam': 8.6})
```

```
LR.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
▾                    LogisticRegression
LogisticRegression(class_weight={'ham': 1.3, 'spam': 8.6})
```

```
predictions = LR.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, classification_report
print(classification_report(y_test, predictions))
```