Data Preprocessing Class

```python
## Author : Ashantha Rosary
import pyspark.sql.functions as F
from pyspark.sql.functions import col, when, regexp_extract, lit, date_sub, to_date, regexp_replace, lower, udf, trim, length
from pyspark.sql.types import StringType
import re
import string
import time
from googletrans import Translator

class DataPreprocessor:
    def __init__(self, dataframe, spark_session):
        self.df = dataframe
        self.spark = spark_session

    def remove_missing_values(self, columns=None):
        if columns:
            self.df = self.df.dropna(subset=columns)
        else:
            self.df = self.df.dropna()
        return self

    def remove_duplicates(self, columns=None):
        if columns:
            self.df = self.df.dropDuplicates(subset=columns)
        else:
            self.df = self.df.dropDuplicates()
        return self

    def convert_relative_dates(self, date_column, reference_date):
        self.df = self.df.withColumn(
            date_column,
            when(col(date_column).rlike(r"(\d+)\s+weeks?\s+ago"),
                 date_sub(to_date(lit(reference_date), 'yyyy-MM-dd'), regexp_extract(col(date_column), r"(\d+)", 1).cast("int") * 7)
                )
            .when(col(date_column).rlike(r"(\d+)\s+days?\s+ago"),
                  date_sub(to_date(lit(reference_date), 'yyyy-MM-dd'), regexp_extract(col(date_column), r"(\d+)", 1).cast("int"))
             )
            .otherwise(col(date_column))
        )

        self.df = self.df.withColumn(
            date_column,
            when(col(date_column).rlike(r"\d{4}-\d{2}-\d{2}"),
                 to_date(col(date_column), 'yyyy-MM-dd'))
            .otherwise(to_date(col(date_column), 'dd MMM yyyy'))
        )
        return self

    def remove_words_with_numbers(self, columns):
        for column in columns:
            self.df = self.df.withColumn(column, regexp_replace(col(column), r'\b\w*\d\w*\b', ''))
        return self

    def convert_to_lowercase(self, columns):
        for column in columns:
            self.df = self.df.withColumn(column, lower(col(column)))
        return self

    def remove_punctuation(self, columns):
        # Pattern to replace underscores between words with space
        underscore_pattern = r'(?<=\w)_(?=\w)'

        # Pattern to remove other punctuation
        punctuation_pattern = r'[^\w\s]'

        for column in columns:
            # First, remove underscores between words
            self.df = self.df.withColumn(column, regexp_replace(col(column), underscore_pattern, ' '))
            # Then, remove other punctuation
            self.df = self.df.withColumn(column, regexp_replace(col(column), punctuation_pattern, ' '))

        return self

    def remove_color_family_words(self, columns):
        # Define the words to remove
        words_to_remove = ['color', 'family']

        # Create a regular expression pattern to match these words
        pattern = '|'.join(words_to_remove)

        for column in columns:
            # Replace the specified words with an empty string
            self.df = self.df.withColumn(column, regexp_replace(col(column), pattern, ''))

            # Remove extra spaces created by the replacement and trim leading/trailing spaces
            self.df = self.df.withColumn(column, trim(regexp_replace(col(column), r'\s+', ' ')))
        return self
    def replace_with_custom_dict(self, column, dict_path):
        custom_dict_df = self.spark.read.csv(dict_path, header=True, inferSchema=True)
        dict_rows = custom_dict_df.collect()
        custom_dict = {row['original']: row['translation'] for row in dict_rows}

        def replace_with_custom_dict_udf(text):
            words = text.split()
            translated_words = [custom_dict.get(word, word) for word in words]
            return ' '.join(translated_words)

        replace_udf = udf(replace_with_custom_dict_udf, StringType())
        self.df = self.df.withColumn(column, replace_udf(col(column)))
        return self

    def translate_column(self, columns):
        def translate_text(text):
            try:
                translator = Translator()
                translation = translator.translate(text, src='auto', dest='en')
                return translation.text
            except Exception as e:
                return str(e)

        translate_udf = udf(translate_text, StringType())
        for column in columns:
            self.df = self.df.withColumn(column, translate_udf(col(column)))
        return self

    def trim_whitespace(self, columns):
        for column in columns:
            self.df = self.df.withColumn(column, trim(col(column)))
        return self
```

```python
    def remove_empty_and_whitespace_rows(self, columns):
        for column in columns:
            self.df = self.df.filter(col(column).isNotNull() & (length(col(column)) > 0))
        return self

    def remove_stop_words(self, column):
        stop_words = set([
            'a', 'an', 'the', 'is', 'in', 'to', 'and', 'of', 'that', 'with', 'for', 'on', 'was', 'as', 'by', 'at', 'it', 'this', 'which', 'or', 'from'
        ])

        def remove_stop_words_udf(text):
            if text:
                words = text.split()
                filtered_words = [word for word in words if word.lower() not in stop_words]
                return ' '.join(filtered_words)
            return text

        remove_stop_words_udf = udf(remove_stop_words_udf, StringType())
        self.df = self.df.withColumn(column, remove_stop_words_udf(col(column)))
        return self

    def get_cleaned_data(self):
        return self.df
```