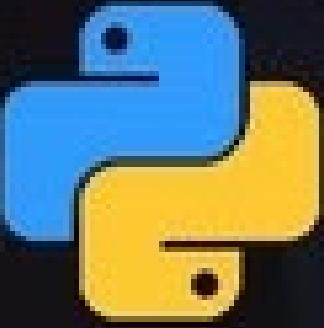




Análisis de Reseñas de Amazon

En este proyecto, exploramos el análisis de reseñas de Amazon, utilizando técnicas de procesamiento de lenguaje natural. El objetivo es extraer información valiosa y útil de los comentarios de los clientes.



python

Preprocesamiento

1 Limpieza de datos

El primer paso consiste en limpiar los datos de reseñas, eliminando caracteres especiales, números y palabras irrelevantes.

2 Tokenización

Se dividen las reseñas en palabras individuales o tokens para su análisis.

3 Eliminación de Stop Words

Se eliminan palabras comunes sin significado contextual, como "el", "la" y "los".

Librerías Necesarias

NLTK (Natural Language Toolkit)

Es una de las bibliotecas más populares en Python para tareas de procesamiento de lenguaje natural.

Herramientas

- Stopwords: Palabras que son comunes y no aportan valor semántico, como "the", "and", "is", etc.
- Tokenizers: Nos permite dividir el texto en palabras (tokens) o en oraciones.

Instalación

Antes de usar NLTK, necesitamos asegurarnos de que está instalada.

```
pip install nltk
```

Data exploration and Data processing

Funciones específicas de NLTK

`nltk.corpus.stopwords`

Contiene conjuntos de palabras que se consideran irrelevantes en muchos análisis de texto. Estas palabras varían por idioma. Usamos el conjunto de palabras en inglés.

`nltk.tokenize.word_tokenize`

Nos permite dividir el texto en palabras individuales.

`nltk.tokenize.sent_tokenize`

Nos permite dividir el texto en oraciones.



Código de Preprocesamiento

```
 1 import nltk
 2 from nltk.corpus import stopwords
 3 from nltk.tokenize import word_tokenize, sent_tokenize
 4
 5 # Asegurarse de que los recursos de NLTK están disponibles
 6 nltk.download('stopwords')
 7 nltk.download('punkt')
 8
 9 def preprocess_text(input_file, output_file):
10     # Leer el archivo de entrada con codificación UTF-8
11     with open(input_file, 'r', encoding='utf-8') as infile:
12         text = infile.read()
13
14     # Tokenización de las oraciones y palabras
15     sentences = sent_tokenize(text)
16     words = word_tokenize(text.lower())
17
18     # Eliminación de palabras basura
19     stop_words = set(stopwords.words('english'))
20     filtered_words = [word for word in words if word.isalpha() and word not in stop_words]
21
22     # Guardar las palabras filtradas en el archivo de salida
23     with open(output_file, 'w', encoding='utf-8') as outfile:
24         outfile.write(' '.join(filtered_words))
25
26     # Definir archivos de entrada y salida
27     input_file = "archivo.txt"
28     output_file = "preprocessed_dataset.txt"
29
30     # Ejecutar la función de preprocesamiento
31     preprocess_text(input_file, output_file)
32
```

Estructura de Preprocesamiento

Antes de usar las funciones, es necesario descargar algunos recursos adicionales de NLTK:

```
import nltk  
nltk.download('stopwords')  
nltk.download('punkt')
```

Tokenización

Luego dividimos el texto en oraciones y palabras:

```
sentences = nltk.sent_tokenize(text)  
words = nltk.word_tokenize(text.lower())
```

Stopwords

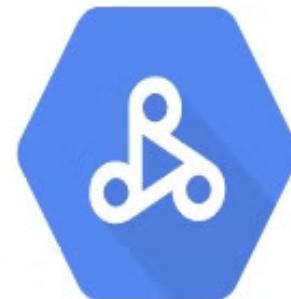
Utilizamos la lista de *stopwords* en inglés de NLTK para filtrar las palabras que no queremos conservar:

```
stop_words = set(nltk.corpus.stopwords.words('english'))  
filtered_words = [word for word in words if word.isalpha() and word not in stop_words]
```

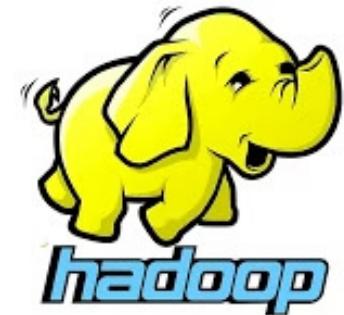
Herramientas para el uso de Hadoop



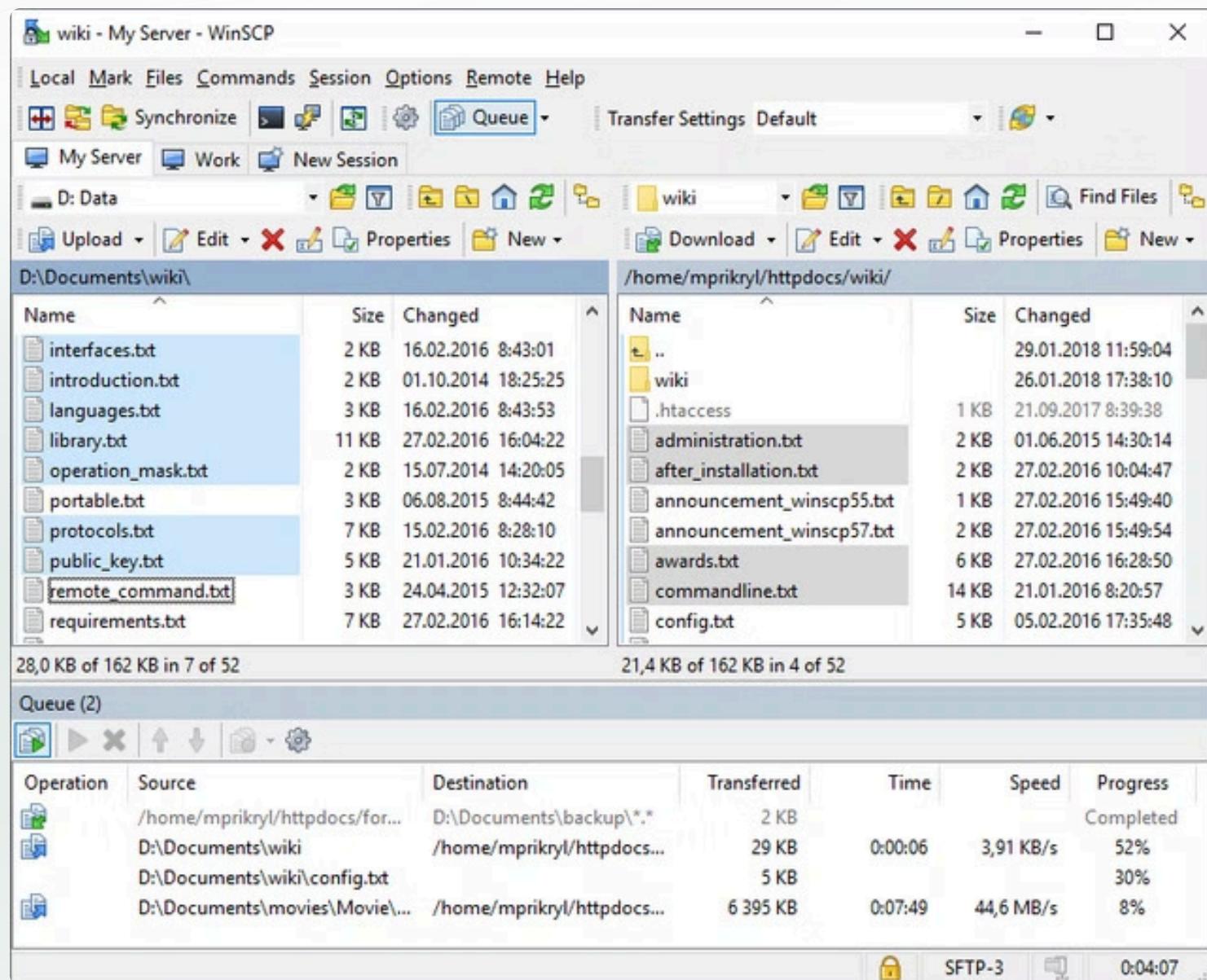
WinSCP
Free FTP CLIENT



Cloud
Dataproc



WinSCP



Pair Mapper

Un **mapper** en Hadoop es una parte clave del modelo de programación **MapReduce**. Su principal función es procesar de manera paralela grandes conjuntos de datos al dividir las tareas en pequeños fragmentos y generar pares clave-valor que luego se agrupan y procesan en la etapa de **reduce**.

WordCount

```
 1 package hadoop_project;
 2
 3 import java.io.IOException;
 4
 5 import org.apache.hadoop.io.IntWritable;
 6 import org.apache.hadoop.io.LongWritable;
 7 import org.apache.hadoop.io.Text;
 8
 9 import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Counter;
11
12 public class M_Mapper extends Mapper<LongWritable, Text, Text, IntWritable> {
13
14     static enum CountersEnum {
15         INPUT_WORDS
16     }
17
18     public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException {
19         String line = value.toString();
20         String[] words = line.split(" ");
21
22         for (String word : words) {
23             Text outputKey = new Text(word.toUpperCase().trim());
24
25             IntWritable outputValue = new IntWritable(1);
26
27             con.write(outputKey, outputValue);
28
29             Counter counter = con.getCounter(CountersEnum.class.getName(),
30                 CountersEnum.INPUT_WORDS.toString());
31             counter.increment(1);
32
33         }
34     }
35
36 }
```

Frequency Analysis

```
● ● ●  
1 package hadoop_project;  
2  
3 import java.io.IOException;  
4 import java.util.Arrays;  
5 import java.util.StringTokenizer;  
6 import org.apache.hadoop.io.IntWritable;  
7 import org.apache.hadoop.io.Text;  
8 import org.apache.hadoop.mapreduce.Mapper;  
9 import javax.naming.Context;  
10  
11 public class M_Mapper extends Mapper<Object, Text, Text, IntWritable> {  
12     private final static IntWritable one = new IntWritable(1);  
13     private Text wordPair = new Text();  
14  
15     public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  
16         StringTokenizer iterator = new StringTokenizer(value.toString());  
17         int numTokens = iterator.countTokens();  
18  
19         if (numTokens < 2) {  
20             return;  
21         }  
22  
23         String[] words = new String[numTokens];  
24  
25         for (int i = 0; iterator.hasMoreTokens(); i++) {  
26             words[i] = iterator.nextToken();  
27         }  
28  
29         Arrays.sort(words);  
30  
31         for (int h = 0; h < numTokens - 1; h++) {  
32             for (int k = h + 1; k < numTokens; k++) {  
33                 if (!words[h].equals(words[k])) {  
34                     wordPair.set(words[h] + " " + words[k]);  
35                     context.write(wordPair, one);  
36                 }  
37             }  
38         }  
39     }  
40 }  
41 }  
42 }
```

Reducer

En esta fase, los pares clave-valor generados por los mappers se agrupan por la clave común. El **reducer** toma todas las claves iguales, realiza una operación de agregación (como sumar, contar, promediar, etc.) y genera una salida final consolidada.

WordCount

```
 1 package hadoop_project;
 2
 3 import java.io.IOException;
 4
 5 import org.apache.hadoop.io.IntWritable;
 6
 7 import org.apache.hadoop.io.Text;
 8
 9 import org.apache.hadoop.mapreduce.Reducer;
10
11 import java.util.Comparator;
12 import java.util.HashMap;
13 import java.util.Map;
14 import java.util.TreeMap;
15
16 import java.util.Iterator;
17
18 public class R_Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {
19
20     HashMap<String, Integer> topWords = new HashMap<String, Integer>();
21
22     public void reduce(Text key, Iterable<IntWritable> values,
23                         Context context) throws IOException, InterruptedException {
24         int sum = 0;
25         for (IntWritable val : values) {
26             sum += val.get();
27         }
28         if (topWords.containsKey(key + "")) {
29             topWords.put(key.toString(), topWords.get(key + "") + sum);
30         } else {
31             topWords.put(key.toString(), sum);
32         }
33     }
34
35     @Override
36     protected void cleanup(Context context) throws IOException, InterruptedException {
37
38         ValueComparator bvc = new ValueComparator(topWords);
39         TreeMap<String, Integer> sorted_map = new TreeMap<String, Integer>(bvc);
40
41         sorted_map.putAll(topWords);
42         Iterator itr = sorted_map.entrySet().iterator();
43         while (itr.hasNext()) {
44             Map.Entry mp = (Map.Entry) itr.next();
45             Text OutPutKey = new Text(mp.getKey() + "");
46             IntWritable OutPutValue = new IntWritable(Integer.parseInt(mp.getValue() + ""));
47             context.write(OutPutKey, OutPutValue);
48         }
49     }
50
51 }
52 }
53
54 class ValueComparator implements Comparator<String> {
55
56     Map<String, Integer> base;
57
58     public ValueComparator(Map<String, Integer> base) {
59         this.base = base;
60     }
61
62     public int compare(String a, String b) {
63         if (base.get(a) >= base.get(b)) {
64             return -1;
65         } else {
66             return 1;
67         }
68     }
69 }
70 }
```

Frequency Analysis

```
1 package hadoop_project;
2
3 import java.io.IOException;
4 import java.util.Map;
5 import java.util.TreeMap;
6 import java.util.Comparator;
7
8 import org.apache.hadoop.io.IntWritable;
9 import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.mapreduce.Reducer;
11
12 public class R_Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {
13     private TreeMap<Text, Integer> countMap = new TreeMap<>();
14
15     public void reduce(Text key, Iterable<IntWritable> values, Context context)
16             throws IOException, InterruptedException {
17         int sum = 0;
18         for (IntWritable val : values) {
19             sum += val.get();
20         }
21         countMap.put(new Text(key), sum);
22     }
23
24     @Override
25     protected void cleanup(Context context) throws IOException, InterruptedException {
26         Map<Text, Integer> sortedMap = sortByValues(countMap);
27         int counter = 0;
28         for (Map.Entry<Text, Integer> entry : sortedMap.entrySet()) {
29             if (counter++ == 20) {
30                 break;
31             }
32             context.write(entry.getKey(), new IntWritable(entry.getValue()));
33         }
34     }
35
36     private static <K, V extends Comparable<V>> Map<K, V> sortByValues(final Map<K, V> map) {
37         Comparator<K> valueComparator = (k1, k2) -> {
38             int compare = map.get(k2).compareTo(map.get(k1));
39             if (compare == 0)
40                 return 1;
41             else
42                 return compare;
43         };
44
45         Map<K, V> sortedByValues = new TreeMap<>(valueComparator);
46         sortedByValues.putAll(map);
47         return sortedByValues;
48     }
49 }
```

Resultados



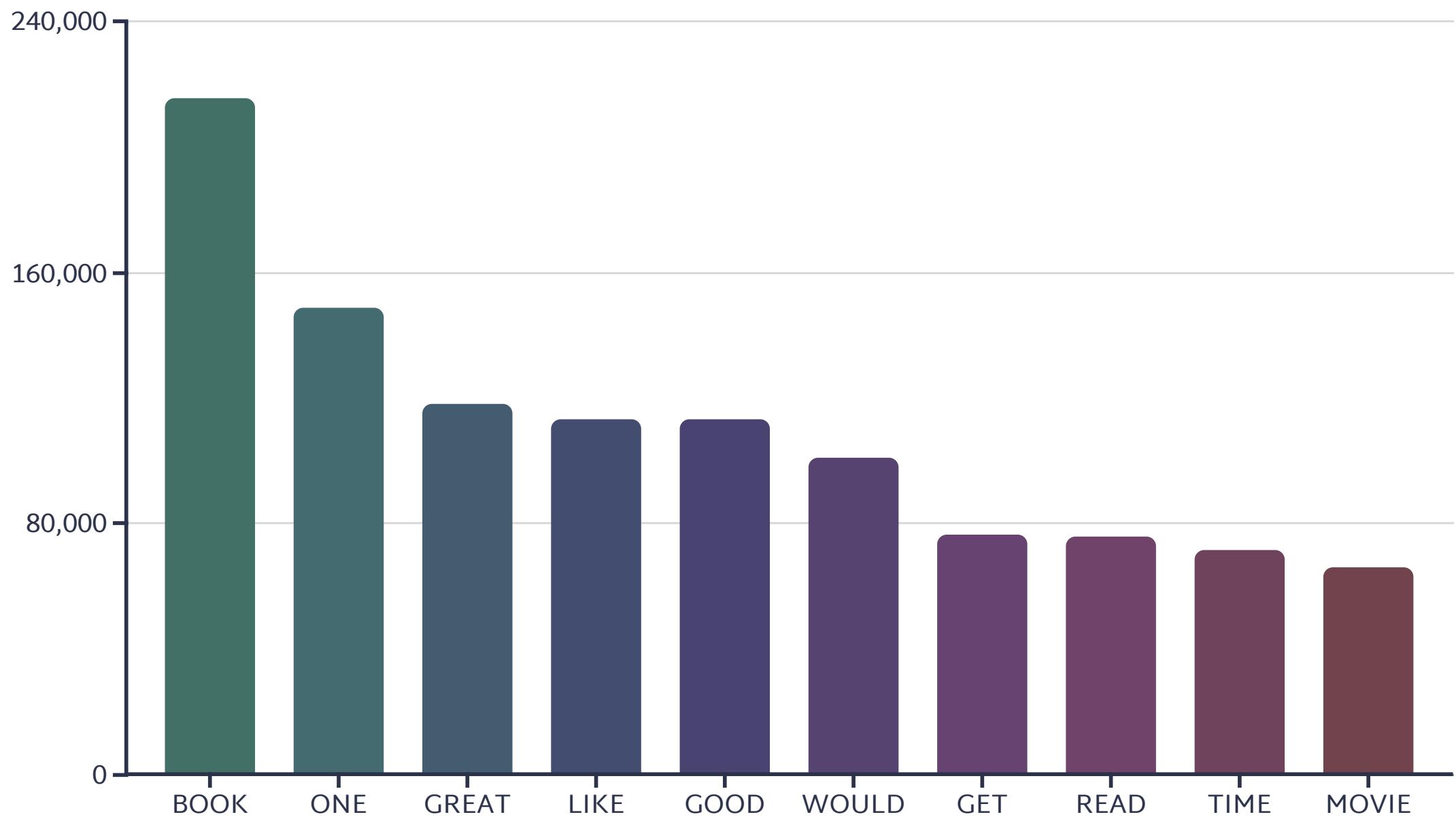
Remazing

"Stuning even for the non-gamer","This sound track was beautiful! It paints the senery in your mind so well I would recomend it even to people who hate vid. game music! I have played the game Chrono Cross but out of all of the games I have ever played it has the best music! ...It would impress anyone who cares to listen! ^_^"

"sizes recomended in the size chart are not real","sizes are much smaller than what is recomended in the chart. I tried to put it and sheer it!. I guess you should not buy this item in the internet..it is better to go to the store and check it"

"one of the last in the series to collect !","The magazine was in very good condition and had the usual high standard of articles and photos that Victoria magazine has come to represent. I have collected all the previous magazines of the series and still enjoy leafing through them on a rainy day"

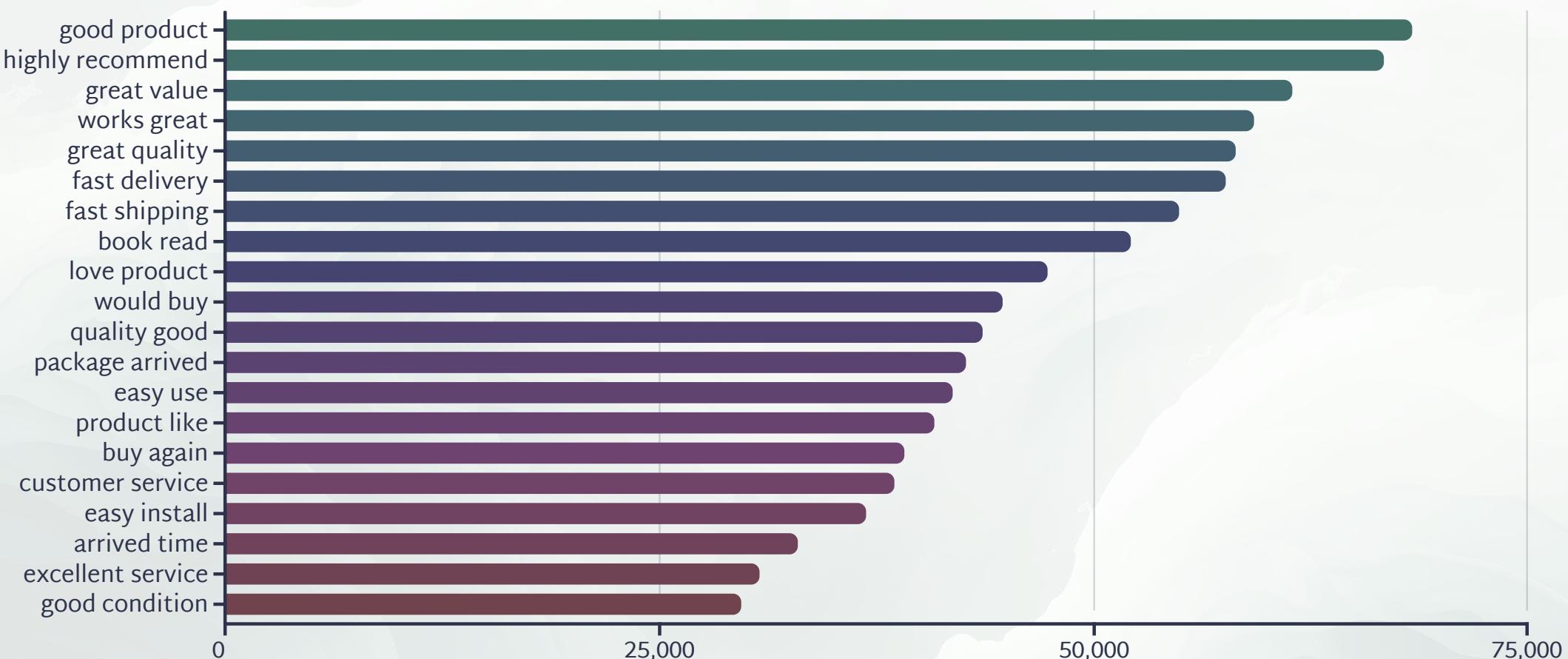
TOP 10 PALABRAS



Hallazgos Interesantes

- ▼ Productos Específicos de Interés
- |
- ▼ Valoración Positiva
- |
- ▼ Recomendaciones y Preferencias de los Clientes

Top 20 itemsets de 2 palabras



▼ Hallazgos Interesantes

- Calidad del producto y satisfacción del Cliente
- Recomendaciones y repeticiones de compra
- Rapidez y Eficiencia
- Facilidad de Uso



Conclusiones



Comprensión del Modelo MapReduce

Entendimos cómo procesar grandes volúmenes de datos eficientemente en entornos distribuidos.

Uso Eficiente de Hadoop

Proceso rápido y distribuido de grandes datos, aprovechando sistemas de archivos distribuidos.

Implementación de Aplicaciones de Minería de Datos

- **WordCount:** Identificó palabras más comunes en 6 millones de reseñas.
- **FrequencyAnalysis:** Detectó patrones de opinión en los clientes.

Fundamento para Análisis Avanzados

Base para futuras técnicas de minería de datos y aprendizaje automático.