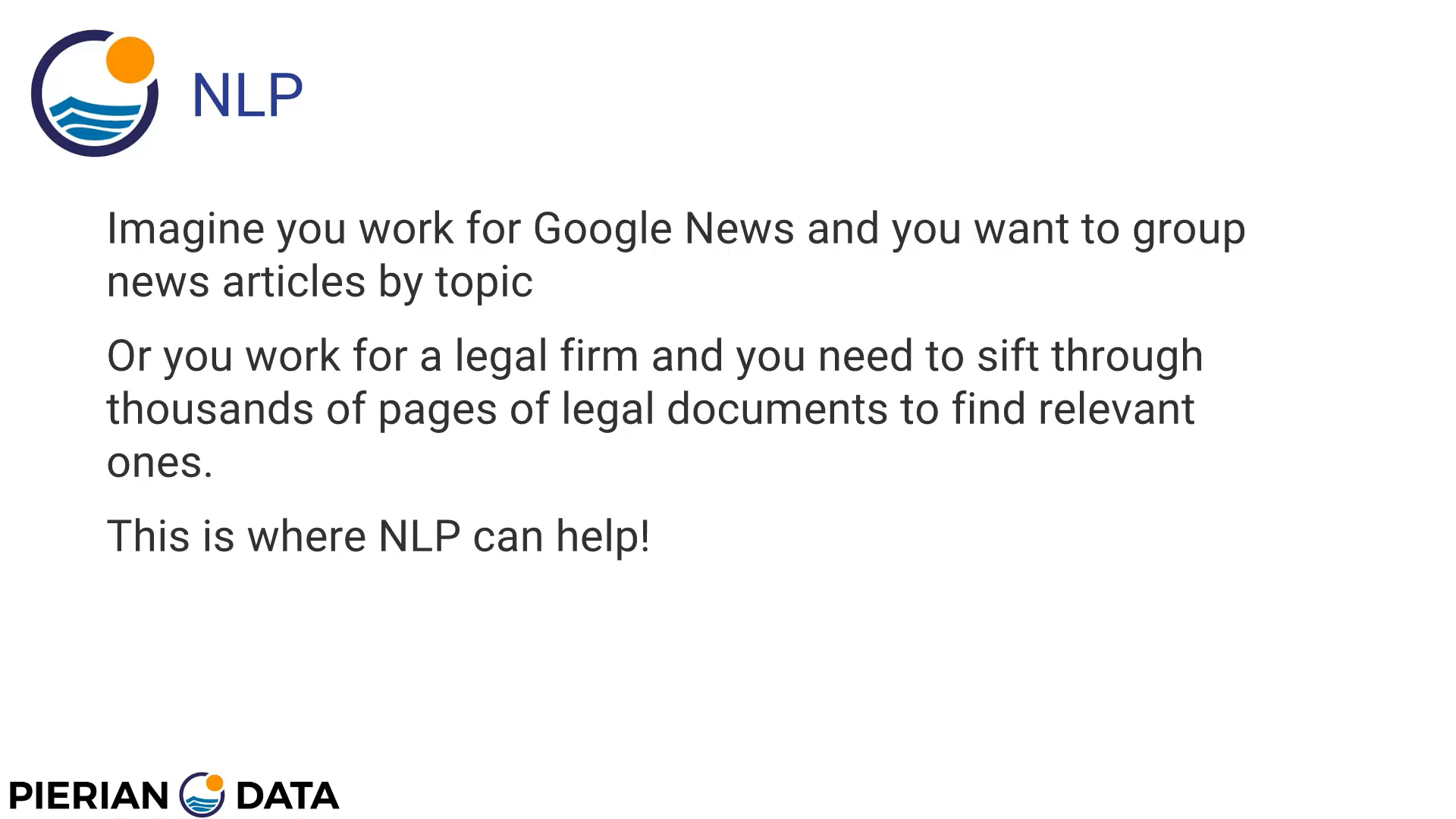
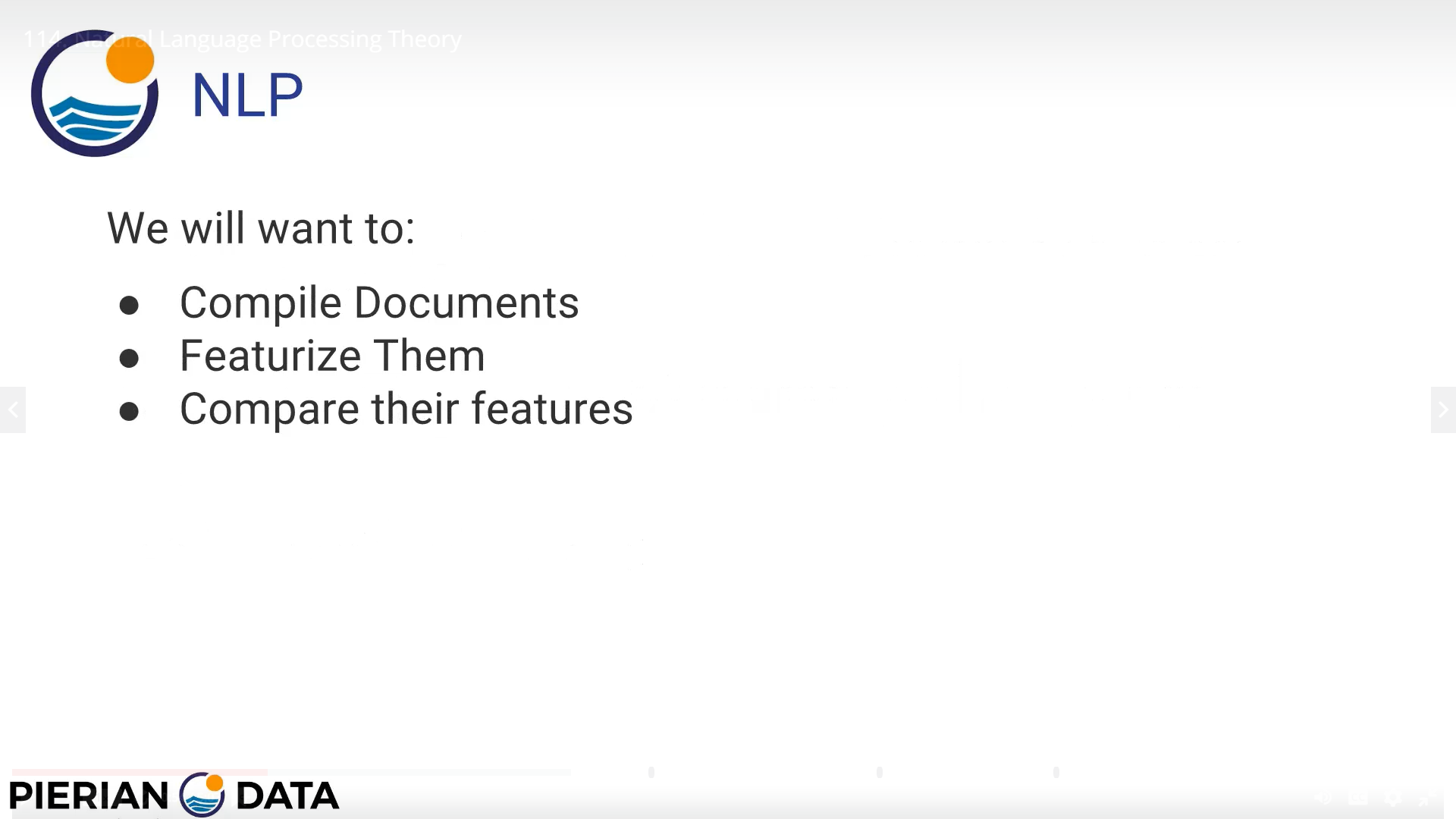
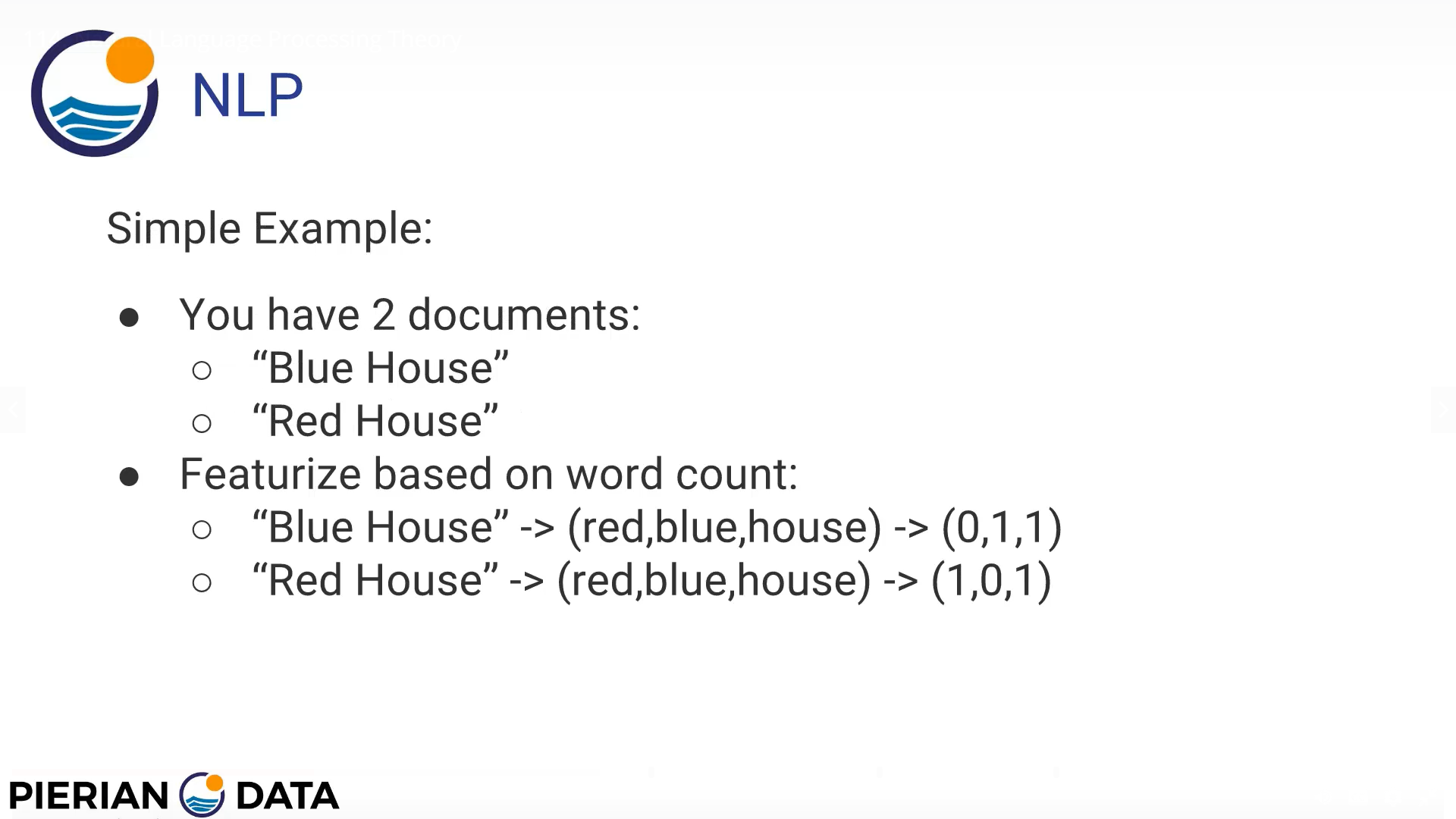
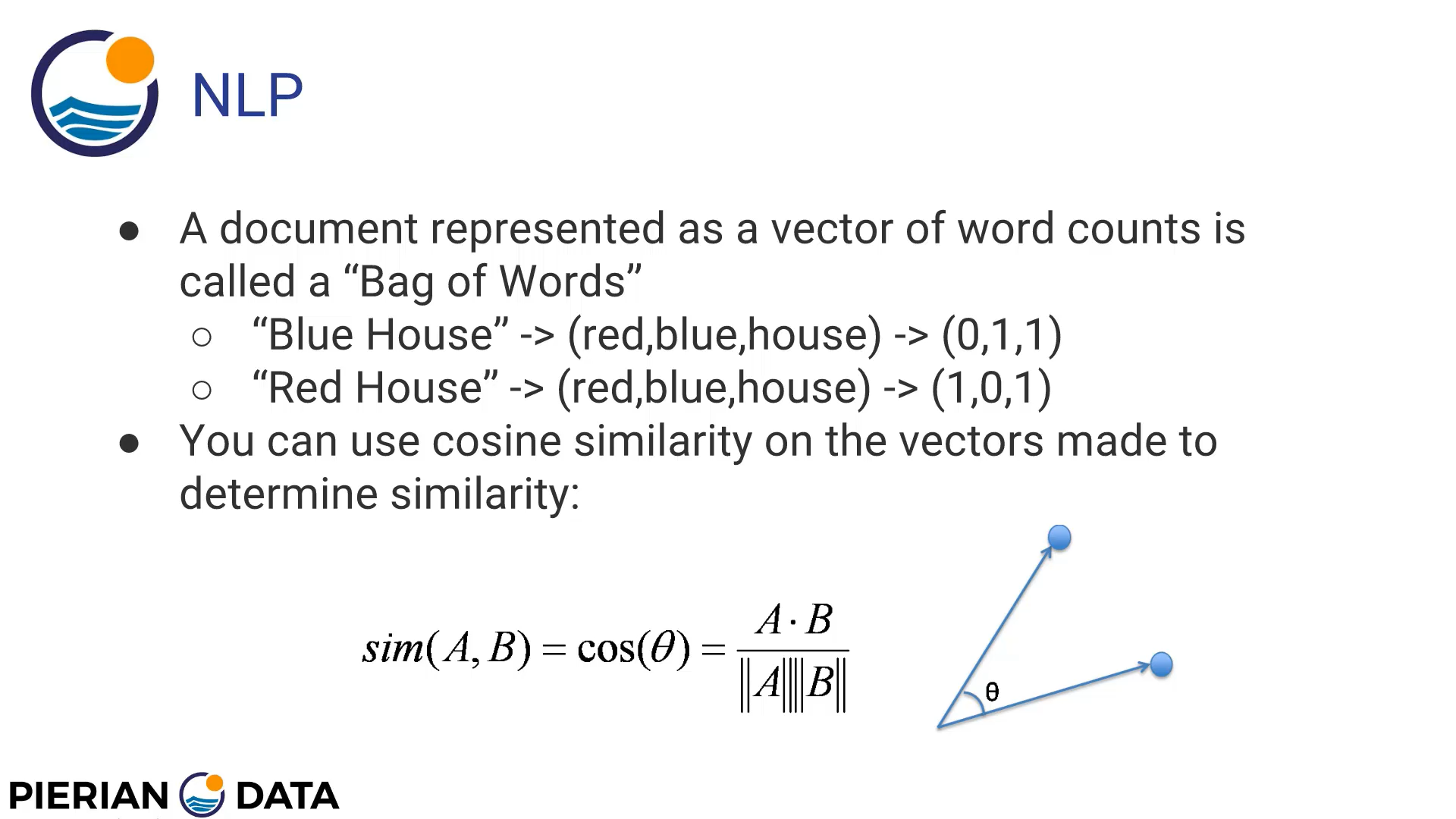
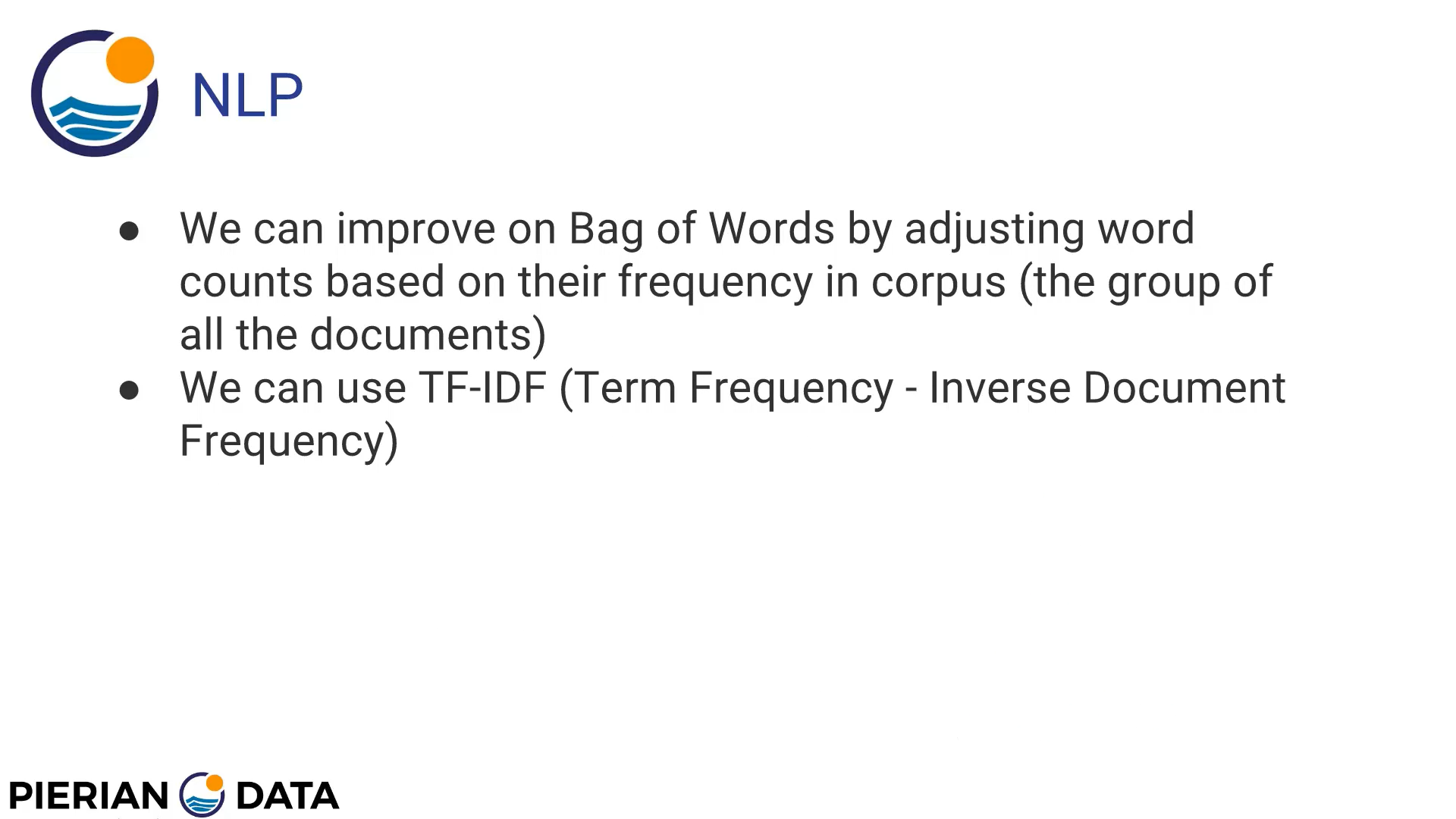
Natural Language Processing

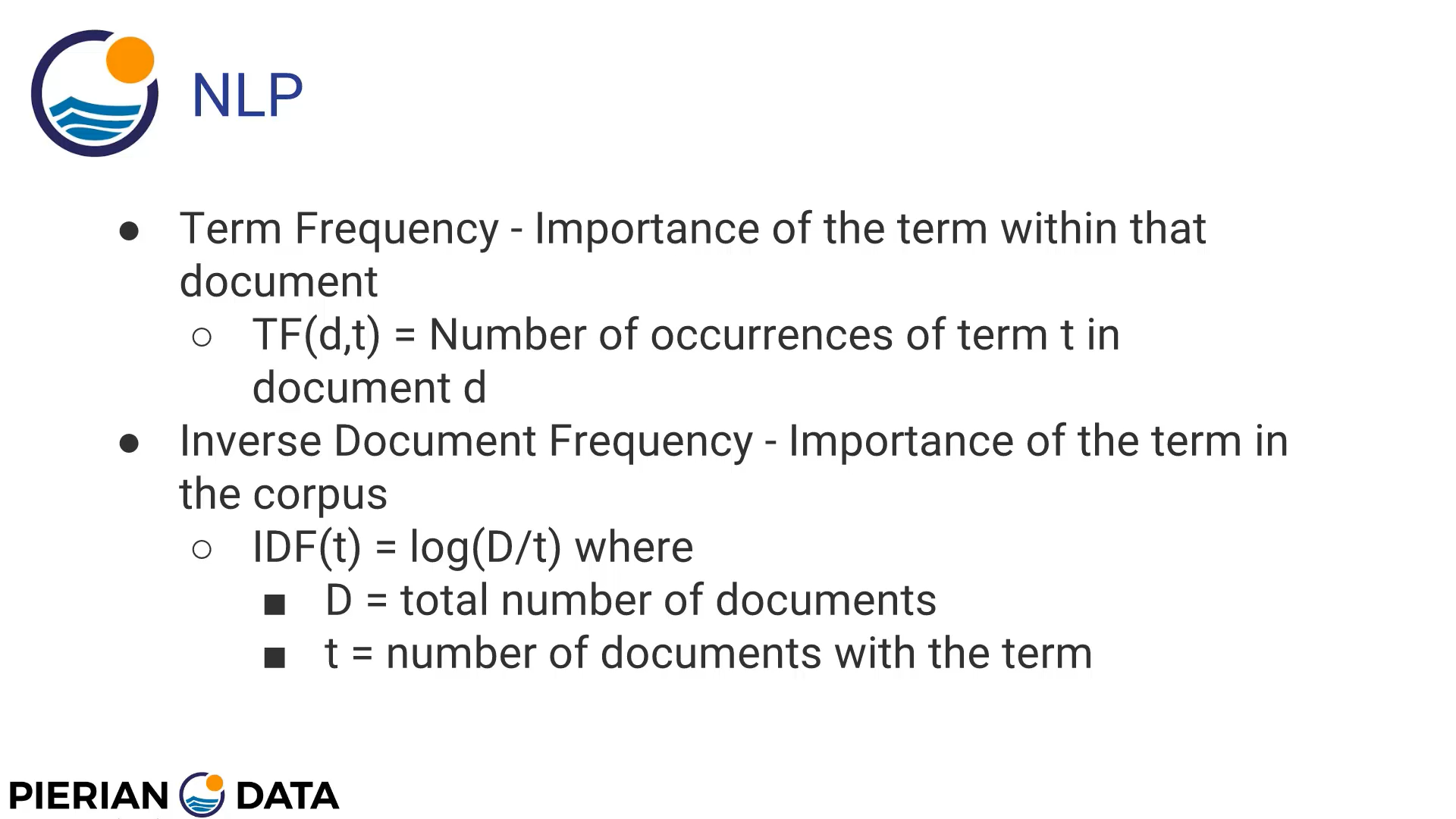


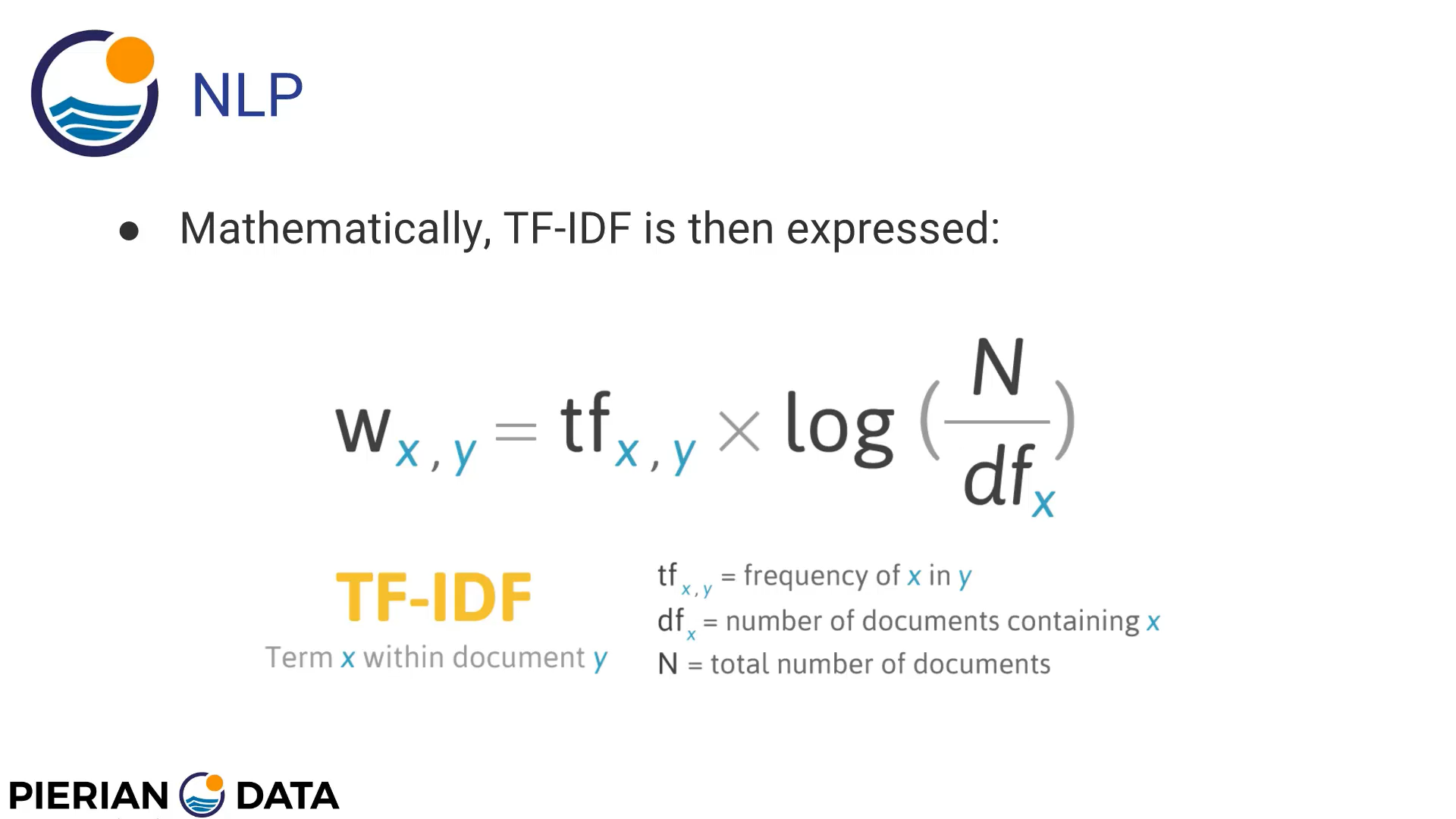












# Downloading required packages from nltk

import nltk  
  
nltk.download\_shell()

we will not comment this download\_shell() command and continue with our code after downloading stopwords corpus data package.

# Importing data from our data file and getting a view of it

import nltk  
  
#nltk.download\_shell()  
  
messages = [line.rstrip() for line in open("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/20-Natural-Language-Processing/smsspamcollection/SMSSpamCollection")]  
print(len(messages))  
for mess\_no,mess in enumerate(messages[:10]):  
 print(str(mess\_no) + " " + mess)

# Creating data frame of our data

messages = pd.read\_csv("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/20-Natural-Language-Processing/smsspamcollection/SMSSpamCollection",sep="\t",names=["label","message"])  
print(messages.head())

# Performing statistical check of data

print(messages.describe())  
print()  
  
print(messages.groupby("label").describe())  
print()

# Creating a length column to store length of message

messages["length"] = messages["message"].apply(len)  
print(messages.head())  
print()

# Doing some data visualisation with length of message

import seaborn as sns  
import matplotlib.pyplot as plt  
  
messages["length"].hist(bins=150)  
print(messages["length"].describe())  
print()  
messages.hist(column="length",by="label",bins=70,figsize=(12,4))

# Pre-processing data

def text\_process(mess):  
 *"""  
 1. remove punc  
 2. remove stop words* ***:param*** *mess: string* ***:return****: return list of clean text words  
 """* nopunc = [char for char in mess if char not in string.punctuation]  
 nopunc = "".join(nopunc)  
 return [word for word in nopunc.split() if word.lower() not in stopwords.words("english")]

# Converting raw sentences into bag of words

import string  
  
from nltk.corpus import stopwords  
print(stopwords.words("english"))  
print()  
  
# Tokenization and defining word vectors  
from sklearn.feature\_extraction.text import CountVectorizer  
bow\_transformer = CountVectorizer(analyzer=text\_process).fit(messages["message"])  
print(len(bow\_transformer.vocabulary\_))  
print()  
  
# Taking one message and checking its word vector  
mess4 = messages["message"][3]  
bow4 = bow\_transformer.tranforms(mess4)  
print(bow4)  
print(bow4.shape)  
print()

# Getting a particular word from bag of words  
print(bow\_transformer.get\_feature\_names()[4068])

# Transforming messages data frame into bag of words vectorisation  
messages\_bow = bow\_transformer.transform(messages["message"])  
print("Shape of Sparse Matrix : " + str(messages\_bow.shape))  
print("Non zero values : " + str(messages\_bow.nnz))  
sparsity = ((messages\_bow.nnz/(messages\_bow.shape[0]\*messages\_bow.shape[1]))\*100)  
print("Sparsity : " + str(sparsity))

# Calculating TF-IDF

# Calculating tf-idf using scikit transformers  
from sklearn.feature\_extraction.text import TfidfTransformer  
tfidf\_transformer = TfidfTransformer().fit(messages\_bow)  
  
# Checking tf-idf of bow4 message  
tfidf4 = tfidf\_transformer.transform(bow4)  
print(tfidf4)  
print()  
  
# We can also check idf of a particular word as  
print(tfidf\_transformer.idf\_[bow\_transformer.vocabulary\_["university"]])  
print()  
  
# Converting our whole bag of words corpus into tf-idf  
messages\_tfidf = tfidf\_transformer.transform(messages\_bow)

# Using Naïve Bayes Classifier to predict spam

# Using Naive-Bayes Classifier to classify spam and ham messages  
from sklearn.naive\_bayes import MultinomialNB  
spam\_detect\_model = MultinomialNB().fit(messages\_tfidf,messages["label"])  
all\_prediction = spam\_detect\_model.predict(messages\_tfidf)  
print(all\_prediction)  
print()  
  
# Checking prediction on single message  
print(spam\_detect\_model.predict(tfidf4)[0])  
print()

# Doing train-test split

# Splitting data into training and testing data  
from sklearn.model\_selection import train\_test\_split  
msg\_train,msg\_test,label\_train,label\_test = train\_test\_split(messages["message"],messages["label"],test\_size=0.3)

# Creating a scikit pipeline so as not to repeat all the process again

# Creating pipeline of transforms  
from sklearn.pipeline import Pipeline  
pipeline = Pipeline([  
 ("bow\_step",CountVectorizer(analyzer=text\_process)),  
 ("tfidf\_step",TfidfTransformer()),  
 ("classifier\_step",MultinomialNB())  
])# our step name can be anything  
pipeline.fit(msg\_train,label\_train)  
predictions = pipeline.predict(msg\_test)

# Printing classification report

# Calculating classification report  
from sklearn.metrics import classification\_report  
print(classification\_report(label\_test,predictions))

# Complete Code –

def text\_process(mess):  
 *"""  
 1. remove punc  
 2. remove stop words* ***:param*** *mess: string* ***:return****: return list of clean text words  
 """* nopunc = [char for char in mess if char not in string.punctuation]  
 nopunc = "".join(nopunc)  
 return [word for word in nopunc.split() if word.lower() not in stopwords.words("english")]  
  
import nltk  
import pandas as pd  
  
#nltk.download\_shell()  
  
# messages = [line.rstrip() for line in open("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/20-Natural-Language-Processing/smsspamcollection/SMSSpamCollection")]  
# print(len(messages))  
# for mess\_no,mess in enumerate(messages[:10]):  
# print(str(mess\_no) + " " + mess)  
  
messages = pd.read\_csv("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/20-Natural-Language-Processing/smsspamcollection/SMSSpamCollection",sep="\t",names=["label","message"])  
print(messages.head())  
print()  
  
print(messages.describe())  
print()  
  
print(messages.groupby("label").describe())  
print()  
  
messages["length"] = messages["message"].apply(len)  
print(messages.head())  
print()  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
messages["length"].hist(bins=150)  
print(messages["length"].describe())  
print()  
messages.hist(column="length",by="label",bins=70,figsize=(12,4))  
  
import string  
  
from nltk.corpus import stopwords  
print(stopwords.words("english"))  
print()  
  
# Tokenization and defining word vectors  
from sklearn.feature\_extraction.text import CountVectorizer  
bow\_transformer = CountVectorizer(analyzer=text\_process).fit(messages["message"])  
print(len(bow\_transformer.vocabulary\_))  
print()  
  
# Taking one message and checking its word vector  
mess4 = messages["message"][3]  
bow4 = bow\_transformer.transform([mess4])  
print(bow4)  
print(bow4.shape)  
print()  
  
# Getting a particular word from bag of words  
print(bow\_transformer.get\_feature\_names()[4068])  
  
# Transforming messages data frame into bag of words vectorisation  
messages\_bow = bow\_transformer.transform(messages["message"])  
print("Shape of Sparse Matrix : " + str(messages\_bow.shape))  
print("Non zero values : " + str(messages\_bow.nnz))  
sparsity = ((messages\_bow.nnz/(messages\_bow.shape[0]\*messages\_bow.shape[1]))\*100)  
print("Sparsity : " + str(sparsity))  
print()  
  
# Calculating tf-idf using scikit transformers  
from sklearn.feature\_extraction.text import TfidfTransformer  
tfidf\_transformer = TfidfTransformer().fit(messages\_bow)  
  
# Checking tf-idf of bow4 message  
tfidf4 = tfidf\_transformer.transform(bow4)  
print(tfidf4)  
print()  
  
# We can also check idf of a particular word as  
print(tfidf\_transformer.idf\_[bow\_transformer.vocabulary\_["university"]])  
print()  
  
# Converting our whole bag of words corpus into tf-idf  
messages\_tfidf = tfidf\_transformer.transform(messages\_bow)  
  
  
# Using Naive-Bayes Classifier to classify spam and ham messages  
from sklearn.naive\_bayes import MultinomialNB  
spam\_detect\_model = MultinomialNB().fit(messages\_tfidf,messages["label"])  
all\_prediction = spam\_detect\_model.predict(messages\_tfidf)  
print(all\_prediction)  
print()  
  
# Checking prediction on single message  
print(spam\_detect\_model.predict(tfidf4)[0])  
print()  
  
# Splitting data into training and testing data  
from sklearn.model\_selection import train\_test\_split  
msg\_train,msg\_test,label\_train,label\_test = train\_test\_split(messages["message"],messages["label"],test\_size=0.3)  
  
  
# Creating pipeline of transforms  
from sklearn.pipeline import Pipeline  
pipeline = Pipeline([  
 ("bow\_step",CountVectorizer(analyzer=text\_process)),  
 ("tfidf\_step",TfidfTransformer()),  
 ("classifier\_step",MultinomialNB())  
])# our step name can be anything  
pipeline.fit(msg\_train,label\_train)  
predictions = pipeline.predict(msg\_test)  
  
# Calculating classification report  
from sklearn.metrics import classification\_report  
print(classification\_report(label\_test,predictions))  
  
plt.show()