CÂU 1

```
import pandas as pd
import numpy as np
import random
import hashlib
```

```
file_path = "WebOfScience-5736.txt"

# Đọc tập dữ liệu và lưu vào một danh sách
with open(file_path, "r", encoding="utf-8") as file:
    dataset = [line.rstrip() for line in file.readlines()]
```

```
total_documents = len(dataset)
```

```
total_documents
```

```
5736
```

```
df = pd.DataFrame(dataset)
df.columns = ['text']
```

```
df
```

	text
0	Phytoplasmas are insect-vectored bacteria that...
1	Background: (-)-alpha-Bisabolol, also known as...
2	A universal feature of the replication of posi...
3	1,2-Dichloropropane (1,2-DCP) and dichlorometh...
4	This paper presents the simulation results of ...
...	...
5731	The intercalation of L-phenylalanate (LP) into...
5732	There is current interest in harnessing the co...
5733	Aim: The zinc finger antiviral protein (ZAP) i...
5734	The present article reviews the biotechnologic...
5735	This paper focuses on a new kind of artificial...

5736 rows × 1 columns

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

class InMemoryMinHashLSH:

    #Ham khoi tao
    def __init__(self, documents):
        self.documents = documents
        self.shingle_set = set()
        self.bool_vectors = None
        self.signatures = None
        self.hash_tables = None
        self.similarities_result = None

    def shingling(self, df):

        #Lua chon k-shingle bang 5
        k = 5

        #Thuc hien shingling gia tri cua key_doc nhap vao

        row = df.iloc[0]['text']
        for i in range(len(row) - k + 1):
            self.shingle_set.add(row[i:i+k])

        # Tao ra mot bool_vector voi row la shingle, column la cac document voi gia tri la 0

        bool_vectors = pd.DataFrame(0, index=range(len(self.shingle_set)),
                                    columns=range(len(df)))

        # Duyet qua tung document
        for s_index, s in enumerate(df['text']):

            # Duyet qua tung shingle
            for e_index, e in enumerate(self.shingle_set):

                #Neu shingle xuat hien trong document
                if e in s:

                    #Gia tri cua bool_vector o vi tri do se bang 1
                    bool_vectors.at[e_index, s_index] = 1

        return bool_vectors


    def minhashing(self, bool_vectors):

        # 100 ham hash
        num_hashes = 100

        # Lay ra row va column cua bool vector
        row_doc, column_doc = bool_vectors.shape


        # Tao ra 100 ham hash ngau nhien
        hash_functions = [hashlib.md5(bytes('hash{}'.format(i), 'utf-8')).digest() for i in range(num_hashes)]

        # Tao ra dataframa signatures voi row la 100 ham hash, column la so luong column cua bool vector
        # Tat ca cac gia tri trong signatures la gia tri vo cuc
        signatures = pd.DataFrame(np.inf, index=range(num_hashes), columns=range(column_doc))

        # Duyet qua hang trong bool
        for r in range(row_doc):
            # Duyet qua cot
            for c in range(column_doc):
                # Neu tai hang va cot do gia tri cua bool_vector bang 1
                if bool_vectors.at[r, c] == 1:

                    # Duyet qua tung vi tri cua ham hash
                    for h in range(len(hash_functions)):
                        # Gia tri nay duoc tinh bang hash_function[h](r) la hash_function thu h roi truyen gia tri r vo tinh de duoc ket qua
                        hash_val = int.from_bytes(hash_functions[h], byteorder='big') ^ r
                        # Neu gia tri nho hon gia tri o signatures
                        if hash_val < signatures.at[h, c]:
                            # Thay the gia tri do bang gia tri hash_val
                            signatures.at[h, c] = hash_val
```

```
        return signatures

    def locality_sensity_hashing(self, signatures_dataframe):


        # Lay ra row cua signature
        num_columns = signatures_dataframe.shape[0]

        # Chia row row ra lam 20 bands
        bands_num = 20
        bands = []

        hash_tables = []

        k = 1000

        # Moi band co 5 hang
        for i in range(0, num_columns, 5):
            band = signatures_dataframe.iloc[i:i+5].to_numpy()
            bands.append(band)

        # Tạo hash table cho từng band
        for band in bands:
            hash_table = {}
            # Moi band co mot ham hash rieng
            hash_function = hashlib.sha256()

            #Thuc hien hash tung column vo hash_table
            # Duyet qua tung colum trong band
            for j, column in enumerate(band.T):


                total_bytes = b''
                total_bytes += str(column).encode('utf-8')
                ascii_string = total_bytes.decode('ascii')

                hash_function.update(ascii_string.encode('utf-8'))

                hash_value = hash_function.hexdigest()

                # Hash cac cot trong band
                bucket = int(hash_value, 16) % k


                # Cac cot giong nhau se duoc cho vao cac hash_table co bucket_id giong nhau
                if bucket not in hash_table:
                    hash_table[bucket] = [j]
                else:
                    hash_table[bucket].append(j)

            hash_tables.append(hash_table)

            # tra ve hash_tables
        return hash_tables

    def run(self):
        # Thực hiện chay va luu tru du lieu vo cac bien
        bool_vectors = self.shingling(self.documents)
        self.bool_vectors = bool_vectors
        signatures = self.minhashing(bool_vectors)
        self.signatures = signatures
        self.hash_tables = self.locality_sensity_hashing(signatures)

    def approxNearestNeightbors(self, key, n):

        # Khởi tạo DataFrame mới từ key
        new_row = pd.DataFrame({'text': [key]})
        # Thêm DataFrame mới vào đầu của documents
        self.documents = pd.concat([new_row, self.documents]).reset_index(drop=True)

        # Gọi hàm run chỉ sau khi cập nhật df
        self.run()

        key_shingles = set()
        k = 5
```

```
        # Tạo tập shingle cho key document
        for i in range(len(key) - k + 1):
            key_shingles.add(key[i:i+k])

        # Tim cac document gan giong trong hash_tables vua tim duoc

        candidate_documents = set()
        for hash_table in self.hash_tables:
            for key, val in hash_table.items():
                if 0 in val:
                    candidate_documents.update(val)

        # Tim do tuong dong trong candidate_documents
        similarities = []
        for doc_index in candidate_documents:
            if doc_index == 0:
                pass
            doc_shingles = set()
            row = self.documents.iloc[doc_index]['text']

            for j in range(len(row) - k + 1):
                doc_shingles.add(row[j:j+k])

            # Su dung jaccard de tinh do tuong dong giua cac document voi nhau giua tren shingles

            similarity = self.jaccard_similarity(key_shingles, doc_shingles)
            similarities.append((doc_index, similarity))

        #Sap xep thu tu cac gia tri similarity giam dan

        similarities.sort(key=lambda x: x[1], reverse=True)

        self.similarities_result = similarities

        # Lay ra n cac gia tri dau tien

        top_documents = [self.documents.iloc[i]['text'] + '\n' for i, _ in similarities[1:n+1]]


        return top_documents

    # Ham jaccard similarity
    def jaccard_similarity(self, set1, set2):
        intersection = len(set1.intersection(set2))
        union = len(set1.union(set2))
        return intersection / union if union != 0 else 0



minhash_lsh = InMemoryMinHashLSH(df)
```

```
# Truy cập kết quả bool_vectors'
key = '''Phytoplasmas, which are bacteria carried by insects, cause diseases in various types of plants. \
Recent advancements in molecular DNA analysis and other methods have resulted in a surge of discoveries regarding phytoplasma-plant relation
and taxonomic classifications. However, the use of common names based on associated diseases, along with different systems for classifying p
complicates understanding. We address this issue by focusing on Australian plant systems, providing a comprehensive overview of phytoplasmas
and diseases. While only certain phytoplasma groups have been identified in Australia, there is a need for ongoing biosecurity measures to p
Many phytoplasmas in Australia remain poorly studied, suggesting the presence of unidentified groups. Some well-studied phytoplasmas infect
further complicating disease management. Additionally, few vectors have been identified for phytoplasmas, particularly in Australia. \
Despite recent progress in molecular research, phytoplasmas remain one of the least understood groups of plant pathogens, posing a significa


bool_vectors_result = minhash_lsh.bool_vectors
result = minhash_lsh.approxNearestNeightbors(key, 5)
# minhash_lsh.similarities_result
result


# # Truy cập kết quả signatures
# signatures_result = minhash_lsh.signatures
# signatures_result

# # Truy cập kết quả hash_tables
# hash_tables_result = minhash_lsh.hash_tables
# hash_tables_result
# print("\nHash Tables:")
# print(hash_tables_result)
```

```
['Phytoplasmas are insect-vectored bacteria that cause disease in a wide range of plant species. The increasing availability of
molecular DNA analyses, expertise and additional methods in recent years has led to a proliferation of discoveries of phytoplasma-
plant host associations and in the numbers of taxonomic groupings for phytoplasmas. The widespread use of common names based on the
diseases with which they are associated, as well as separate phenetic and taxonomic systems for classifying phytoplasmas based on
variation at the 16S rRNA-encoding gene, complicates interpretation of the literature. We explore this issue and related trends
through a focus on Australian pathosystems, providing the first comprehensive compilation of information for this continent,
covering the phytoplasmas, host plants, vectors and diseases. Of the 33 16Sr groups reported internationally, only groups I, II,
III, X, XI and XII have been recorded in Australia and this highlights the need for ongoing biosecurity measures to prevent the
introduction of additional pathogen groups. Many of the phytoplasmas reported in Australia have not been sufficiently well studied
to assign them to 16Sr groups so it is likely that unrecognized groups and sub-groups are present. Wide host plant ranges are
apparent among well studied phytoplasmas, with multiple crop and non-crop species infected by some. Disease management is further
complicated by the fact that putative vectors have been identified for few phytoplasmas, especially in Australia. Despite rapid
progress in recent years using molecular approaches, phytoplasmas remain the least well studied group of plant pathogens, making
them a "crouching tiger" disease threat.\n',
 'Objective: Root resorption is a complication of orthodontic treatment and till date, there is a dearth of information regarding
this issue. The aim of this study was to determine whether the expression of transforming growth factor-beta 1 (TGF-beta 1, an
inflammatory cytokine) is related to orthodontic force. Moreover, if associated, the expression level may be helpful in differential
diagnosis, control and ultimate treatment of the disease. Materials and Methods: In this experimental study, a total of 24 eight-
week-old male Wistar rats were selected randomly. On day 0, an orthodontic appliance, which consisted of a closed coil spring, was
ligated to the upper right first molar and incisor. The upper left first molar in these animals was not placed under orthodontic
force, thus serving as the control group. On day 21, after anesthesia, the animals were sacrificed. The rats were then divided into
two equal groups where the first group was subjected to histological evaluation and the second group to reverse transcriptase-
polymerase chain reaction (RT-PCR). Orthodontic tooth movement was measured in both groups to determine the influence of the applied
force. Results: Statistical analysis of data showed a significant root resorption between the experimental group and control group
(P<0.05), however, there was no significant difference in the expression level of the inflammatory cytokine, TGF-beta 1. Conclusion:
Based on the findings of this study, we suggest that there is a direct relationship between orthodontic force and orthodontic
induced inflammatory root resorption. In addition, no relationship is likely to exist between root resorption and TGF-beta 1
expression in the resorptive lacunae.\n',
 'Anger in the context of psychosis has a significant impact on treatment outcomes and serious implications for risk management.
Understanding mechanisms underlying anger will improve interventions and inform strategies for prevention. This study is the first
to examine the relationships between anger and key theoretical drivers across different phases of the psychosis continuum. A battery
including measures of theory of mind, attachment, hostile attribution bias, paranoia and anger was administered to 174 participants
(14 ultra-high risk, 20 first-episode, 20 established psychosis, 120 non-clinical participants). We tested the model that insecure
attachment, paranoia, impaired theory of mind and hostile attribution bias would predict trait anger using multiple regression.
Attachment avoidance, paranoia and hostile attribution bias were significantly associated with anger but attachment anxiety and
theory of mind were not. Mediation analysis showed that paranoia partially mediated the relationship between avoidant attachment and
anger but hostile attribution bias did not. Findings emphasise the importance of interventions targeting paranoia to reduce anger
and the potential of preventive strategies focused on attachment relationships in early life or adulthood to reduce adult paranoia
and anger.\n',
 "Rural electrification rates in India lag behind government goals, in part due to the inability of distribution companies (discoms)
to fund central grid expansion. In the absence of central grid electrification, mini-grids offer significant potential for an
immediate pathway toward rural electrification and the attendant gains in economic growth and productivity. Yet private investment
in mini-grids has been virtually absent in India. Using a comprehensive life-cycle cost analysis, we find that mini-grids based on
solar PV power and storage are more economical than incumbent energy services available to households without central grid
connection. Under current law, a prospective entrepreneur in India does not require a license or certification in order to build a
mini-grid and subsequently provide electricity services in the area covered by said installation. Conversely, there is no legal or
regulatory framework that specifies what is to happen if the central grid were to be extended to an area that is already covered by
a mini-grid. We report detailed survey evidence from interviews with entrepreneurs, analysts and policymakers whose assessments
converge on the same point: mini-grid investments would be jeopardized in the event of central grid extension, precisely because
discoms would, by regulatory order, provide electricity services at highly subsidized rates, well below their full economic cost.
Our fieldwork suggests that the threat of central grid extension is a gateway barrier preventing mini-grid development in India. The
issues associated with the gateway barrier have common elements with the so-called holdup problem as identified in the economics of
organizations. There have been two recent federal policy guidelines and one actual state-level policy addressing the regulatory
status of mini-grids. We examine the effectiveness of these policies/proposals in terms of an entrepreneur's willingness to develop
```

```
mini-grids in the future. (C) 2016 Elsevier Ltd. All rights reserved.\n",
 'Aims: Evidence suggests ongoing practice variability in the quality of skeletal survey examinations for non-accidental injury. The purpose of the study was to investigate the effects on examination quality following the implementation of imaging checklists. Method: A retrospective evaluation of skeletal survey examinations was carried out on studies performed between January 2007 and
```

```
minhash_lsh.similarities_result
```

```
[(0, 1.0),
 (1, 0.2986635676932016),
 (5523, 0.08013765978367748),
 (5433, 0.08011527377521614),
 (3995, 0.07970702283498492),
 (5151, 0.0791044776119403),
 (1179, 0.07529654461062403),
 (613, 0.07384169884169885),
 (925, 0.07371601208459215),
 (975, 0.07332205301748448),
 (4209, 0.071849234393404),
 (60, 0.07125307125307126),
 (695, 0.0707070707070707),
 (5471, 0.07047146401985112),
 (2092, 0.07014125669751584),
 (2226, 0.06956521739130435),
 (3225, 0.06935758953951109),
 (3191, 0.06906614785992218),
 (4432, 0.0675990675990676),
 (5506, 0.06719924812030076),
 (732, 0.06643356643356643),
 (143, 0.0659846547314578),
 (724, 0.06565382528486163),
 (1593, 0.06517311608961303),
 (2927, 0.06503251625812906),
 (2807, 0.06316348195329087),
 (351, 0.06275100401606426),
 (3081, 0.0626808100289296),
 (249, 0.06176305446378439),
 (2684, 0.0616332819722650 24),
 (21, 0.061621621621621624),
 (1104, 0.06159238858287431),
 (5481, 0.0615),
 (5198, 0.061422413793103446),
 (1803, 0.06129807692307692),
 (3477, 0.060206422018348624),
 (2717, 0.059654631083202514),
 (4627, 0.05938697318007663),
 (1303, 0.05930568948891032),
 (2300, 0.05873715124816446),
 (1311, 0.0584144645340751),
 (3367, 0.057946314443971025),
 (4398, 0.05788423153692615),
 (4410, 0.057759919638372674),
 (723, 0.05761500669941939),
 (810, 0.05686695278969957),
 (210, 0.05654761904761905),
 (2529, 0.056210335448776065),
 (2030, 0.056074766355140186),
 (4743, 0.05588526211671612),
 (1390, 0.05588393336915637),
 (2506, 0.05563835072031793),
 (2893, 0.05542168674698795),
 (4720, 0.05531505531505532),
 (4752, 0.05487480021310602),
 (5332, 0.05452674897119342),
 (5607, 0.05448408871745419),
 (136, 0.0540641312453393),
```

CÂU 2

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
```

```
from pyspark.sql import SparkSession, Row
from pyspark.ml.feature import MinHashLSH, Tokenizer, NGram, MinHashLSHModel
from pyspark.ml.linalg import Vectors
from pyspark.sql.functions import col, udf
from pyspark.sql.types import ArrayType, StringType, IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.functions import lit
import numpy as np
from pyspark.sql.functions import size
import hashlib
from pyspark.sql import functions as F
from pyspark.sql import Window
from pyspark.sql.functions import posexplode, col, collect_list


def create_char_shingles(text, n=2):
    """ Hàm tạo shingles ký tự từ văn bản. """
    return [text[i:i+n] for i in range(len(text)-n+1)]
def shingles_to_vector(shingles, shingles_list):
    """ Hàm chuyển danh sách shingles thành vector boolean. """
    return [1 if shingle in shingles else 0 for shingle in shingles_list]
def display_arr(arr):
  for i in range (0, len(arr)):
    print(arr[i])
    print("\n")
def jaccard_dis(a, b):
    # Tính toán kích thước của tập hợp giao nhau và tập hợp hợp nhất
    intersection_size = sum(1 for x, y in zip(a, b) if x == y)
    union_size = len(a)

    # Tính toán phần trăm giống nhau dựa trên kích thước tập hợp giao nhau và tập hợp hợp nhất
    if union_size == 0:
        return 0
    else:
        similarity_percentage = intersection_size / union_size
        return similarity_percentage



class InMemoryMinHashLSH:
    def __init__(self, documents):
        self.documents = documents
        self.key_doc = ""
        self.key_bool = []
        self.all_singling = []

    def shingling(self, documents):


        char_shingles_udf = udf(create_char_shingles, ArrayType(StringType()))
        # Áp dụng UDF để tạo shingles ký tự

        shingle_data = self.documents.withColumn("shingles", char_shingles_udf(col("document")))

        # Lấy danh sách tất cả các shingle từ DataFrame đã được tạo
        all_shingles = set(shingle_data.select("shingles").rdd.flatMap(lambda x: x[0]).collect())


        all_shingles_list = list(all_shingles)

        self.all_singling = all_shingles_list
        vector_udf = udf(lambda x: shingles_to_vector(x, all_shingles_list), ArrayType(IntegerType()))

        # Áp dụng UDF để tạo vector boolean
        boolean_vector_df = shingle_data.withColumn("boolean_vector", vector_udf(col("shingles")))

        only_vector_df = boolean_vector_df.select("boolean_vector")

        vector_cols = [col("boolean_vector")[i].alias(f"{i+1}") for i in range(len(all_shingles_list))]
        separated_df = boolean_vector_df.select(*vector_cols)



        return only_vector_df


    def minhashing(self, bool_vectors):
```

```
    num_hashes = 100
    matrix = bool_vectors.select("boolean_vector").collect()

    row_doc = len(matrix)
    column_doc = len(matrix[0][0])

    # Generate 100 hash functions

    hash_functions = [hashlib.md5(bytes('hash{}'.format(i), 'utf-8')).digest() for i in range(num_hashes)]

    # Create signature data frame with infinite value
    # inf_array = np.full((num_hashes, column_doc), float('inf'))
    inf_array = [[float('inf') for _ in range(row_doc)] for _ in range(num_hashes)]
    #signatures = pd.DataFrame(np.inf, index=range(num_hashes), columns=range(row_doc))

    for c in range(column_doc):
      for r in range(row_doc):

            if matrix[r][0][c] == 1:
                for h in range(len(hash_functions)):
                    hash_val = int.from_bytes(hash_functions[h], byteorder='big') ^ c
                    # if hash_val < signatures.at[h,r]:
                    #      signatures.at[h,r] = hash_val
                    if hash_val < inf_array[h][r]:
                          inf_array[h][r]= hash_val


    # Tạo DataFrame từ danh sách các hàng

    return inf_array



def locality_sensitivity_hashing(self, signatures):
    num_columns = len(signatures)

    k = 100

    bands_num = 20
    band = []
    bands = []
    hash_tables = []
    column_buckets = {i: [] for i in range(num_columns)}
    # Tạo band
    for i in range(0, num_columns,5):
        band = signatures[i:i+5]
        bands.append(band)

    for band in bands:
        hash_table = {}
        hash_function = hashlib.sha256()
        a = np.array(band)
        bucket_mapping = {}  # Mapping from hashed_integer to new sequential bucket_id
        next_bucket_id = 0
        temp = []
        bucket_id = 0
        #ham dict key: buck_id, value: column
        for j, column in enumerate(a.T):
            total_bytes = b''
            total_bytes += str(column).encode('utf-8')
            ascii_string = total_bytes.decode('ascii')
            hash_object = hashlib.sha256()


            hash_object.update(ascii_string.encode('utf-8'))

            # Lấy giá trị băm (hash) dưới dạng một chuỗi hexa
            hashed_hex = hash_object.hexdigest()

            hashed_integer = int(hashed_hex, 16)


            # Check if this hash already has a bucket

            bucket = hashed_integer / k

            if bucket not in bucket_mapping:
```

```
                if bucket not in bucket_mapping:
                  bucket_mapping[bucket] = next_bucket_id
                  next_bucket_id += 1

            # Get the new bucket_id and assign the column
                new_bucket_id = bucket_mapping[bucket]
                if new_bucket_id not in hash_table:
                    hash_table[new_bucket_id] = [j]
                else:
                    hash_table[new_bucket_id].append(j)

                total_bytes = b''


            hash_tables.append(hash_table)


        return hash_tables

    def run(self):
        # Complete end-to-end run
        shingle_data = self.shingling(self.documents)
        minhash = self.minhashing(shingle_data)
        lsh = self.locality_sensitivity_hashing(minhash)
        shingle_data.show()
        print(minhash)
        print(lsh)


    def approxNearestNeighbors(self, key, n):

        keys = create_char_shingles(key)
        bool_vector = shingles_to_vector(keys,self.all_singling)
        list_bool = []
        list_bool.append(bool_vector)
```