SCOPUS DATA

PROJECT

WARITSARA CHANTHADET 6638212921
CHIRATT CHIRARATTANACHAN 6638032221
PAVEEPAT YONGAVINSAKUL 6638121421

OUR GOAL

Study the correlation between institution types, geographic locations, and the diversity of topics to determine if certain institutions contribute more to specific areas.

CONTENT

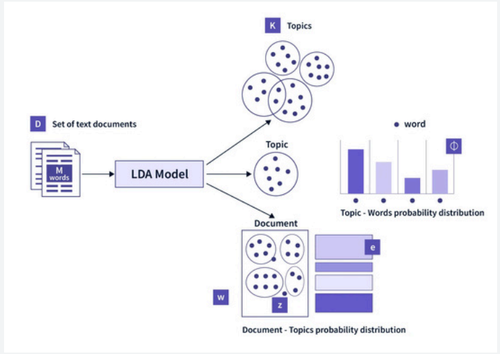
WORKFLOW DIAGRAM

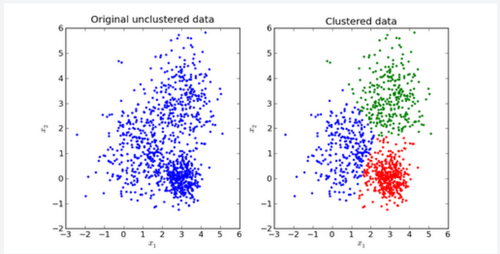
Model Fitting

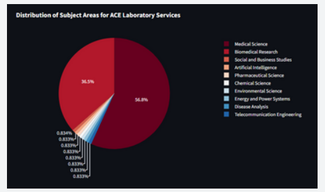
IEEE Xplore® Digital Library

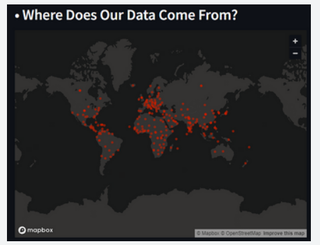
Scopus

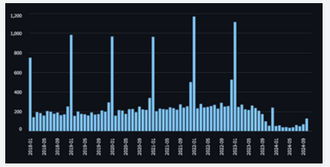
Data











Visualization

DATA PREPERATION

- Data Extraction
- Data Cleaning and Preprocessing Pipeline
- Remove Duplicates
- Format Countries and Institution Names

1) EXTRACT RELEVENT DATA

Raw Data Set

- title
- abstract
- abstract_word_count
- authors
- published date
- affiliations
- keywords
- source_id
- year
- prism:isbn
- language
- citedby_count
- publisher

Data that we’re are interested in

- Title
- Authors
- Published Date
- Institution
- City
- Country
- Keywords

2) DATA CLEANING AND PREPROCESSING PIPELINE

Institution Cleaning

Clean Institution Names

```python
def clean_institution_names(self, df):
    """Clean institution names, remove 'not available' and NaNs."""
    df["Institution"] = df["Institution"].apply(
        lambda x: (
            [inst for inst in x if inst.lower() != "not available"]
            if isinstance(x, list)
            else x
        )
    )
    df["Institution"] = df["Institution"].apply(
        lambda x: (
            [inst for inst in x if pd.notna(inst)] if isinstance(x, list) else x
        )
    )
    return df
```

Removes “Not available” and null values from the Institution Column

Create Author to Institution Mapping

```python
def create_author_to_institution_mapping(self, df):
    """Create a mapping of authors to their known institutions."""
    author_to_institution = {}
    for _, row in df.iterrows():
        if isinstance(row["Authors"], list) and isinstance(
            row["Institution"], list
        ):
            for author in row["Authors"]:
                author_to_institution.setdefault(author, set()).update(
                    row["Institution"]
                )
    return author_to_institution
```

Maps authors to their associated institutions.

2) DATA CLEANING AND PREPROCESSING PIPELINE

Institution Cleaning

Impute Institution

```python
def impute_institution(self, row, author_to_institution, most_common_institution):
    """Impute missing institutions for authors."""
    if not isinstance(row["Institution"], list) or len(row["Institution"]) == 0:
        institutions = set()
        if isinstance(row["Authors"], list):
            for author in row["Authors"]:
                if author in author_to_institution:
                    institutions.update(author_to_institution[author])
        if institutions:
            return list(institutions)
        return [most_common_institution]
    return row["Institution"]
```

Fills missing institutions using the mapping or the most common institution

Date Imputation

```python
def impute_missing_dates(self, df):
    """Impute missing dates with the most common date."""
    most_common_date = (
        df["Date"].mode()[0] if not df["Date"].isnull().all() else None
    )
    df["Date"] = df["Date"].fillna(most_common_date)
    return df
```

Replaces missing dates with the most common date

2) DATA CLEANING AND PREPROCESSING PIPELINE

Keyword Cleaning

Clean and Impute Missing Keywords

```python
def clean_keywords(self, df):
    """Clean and impute missing keywords."""
    vectorizer = CountVectorizer(stop_words="english", max_features=100)
    title_vectors = vectorizer.fit_transform(df["Title"].fillna("No Title"))
    similarity_matrix = cosine_similarity(title_vectors)

    df["Keywords"] = df["Keywords"].apply(
        lambda x: [kw for kw in x if pd.notna(kw)] if isinstance(x, list) else x
    )
    df["Keywords"] = df.apply(
        lambda row: self.infer_keywords(row, df, similarity_matrix), axis=1
    )

    return df
```

Infer Missing Keywords using Natural Language Processing (NLP) techniques

We compute the cosine similarity between the title vectors and replaced the missing keywords based on the title similarity

2) DATA CLEANING AND PREPROCESSING PIPELINE

Location Cleaning

Clean Location

```python
def clean_location(self, df, column_name):
    """Clean city and country columns."""
    df[column_name] = df[column_name].apply(
        lambda x: (
            [item if item.lower() != "not available" else np.nan for item in x]
            if isinstance(x, list)
            else x
        )
    )
    df[column_name] = df[column_name].apply(
        lambda x: (
            [item for item in x if pd.notna(item)] if isinstance(x, list) else x
        )
    )
    return df
```

Removes "not available" and null values for Country Column

Format country's name to standardized name

```python
def clean_country(self, country_name):
    """Clean and standardize country names."""
    country_name = country_name.strip().lower()
    valid_country_names = [country.name.lower() for country in pycountry.countries]
    if country_name:
        match = process.extractOne(country_name, valid_country_names, score_cutoff=80)
        return match[0].title() if match else "Unknown"
    return "Unknown"
```

U S A
U.S.A
↓
United State

1) Retrieve all valid country names from **pycountry** library

2) Using **fuzzzywuzzy** library, we analyze if the name of the country matches any country from the pycountry library.

If the match score if >80 we return the matched (standardized) country name

2) DATA CLEANING AND PREPROCESSING PIPELINE

Expand the Country and Institution Columns

	Title	Authors	Institution	City	Country	Keywords	Date
0	Fast Sampling of Synthetic Apertures in Short ...	['Jonas Schorlemer', 'Aman Batra', 'Thomas Kai...	['Ruhr University Bochum', 'Technische Univers...	['Dresden', 'Bochum', 'Duisburg']	['Germany']	['Random Sampling', 'Radar Sensor', 'Range Com...	2023-01-09

	Title	Authors	Institution	City	Country	Keywords	Date
0	Fast Sampling of Synthetic Apertures in Short ...	['Jonas Schorlemer', 'Aman Batra', 'Thomas Kai...	Ruhr University Bochum	['Dresden', 'Bochum', 'Duisburg']	Germany	['Random Sampling', 'Radar Sensor', 'Range Com...	2023-01-09
1	Fast Sampling of Synthetic Apertures in Short ...	['Jonas Schorlemer', 'Aman Batra', 'Thomas Kai...	Technische Universität Dresden	['Dresden', 'Bochum', 'Duisburg']	Germany	['Random Sampling', 'Radar Sensor', 'Range Com...	2023-01-09

Since in some rows, there may be multiple institutions or countries we want to seperate them into individual rows so we can properly analyze each institution

WEB SCRAPING

- We had choices between using an API to directly call for data or trying web scraping.
- API has its limit on whether to call certain data or times it can be used
- Web scraping on the other hand was very slow.

In IEEExplore, you can change the document ID at the end of the URL

https://ieeexplore.ieee.org/document/{doc_id}

IEEEXPLORE

From that, we can use Selenium to scrape 1000 documents for the title, author, date of publication, institution, country, and city

WEB_SCRAPING.PY

To make sure no data is lost if the process is terminated, we saved the files to {doc_id}.json. This means we had to join all 1000 files to be easily kept.

JOIN_JSON.PY

INSIDE WEB_SCRAPING.PY

```python
# Generate a list of evenly spaced values from Start to End with Num_iterations steps.
def calculate_loop(self):
    step = (self.end - self.start) // self.num_iterations
    self.doc_id_list = [self.start + i * step for i in range(self.num_iterations)]
    self.doc_id_list = self.doc_id_list[::-1]

# Create link to be turned into soups
def create_url(self, doc_id):
    return f"https://ieeexplore.ieee.org/document/{doc_id}"
```

The first function creates a list of doc_id for us to visit and scrape the data.

The second function uses the doc_id to generate the URL.

PREREQUISITES

INSIDE WEB_SCRAPING.PY

The first function will press ok when the site prompts us to click a cookie so the script doesn't break.

The second function is the scraping part. It targets specific tags and retrieves the data

SCRAPING

```python
def click_cookies(self, browser):
        try:
            consent_button = WebDriverWait(browser, self.delay).until(
                EC.element_to_be_clickable((By.ID, "cookieConsentButtonID"))
            )
            consent_button.click()
        except:
            # print("Cookie consent button not found or not needed.")
            pass

    def find_title(self, browser):
        try:
            title_element = browser.find_element(
                By.XPATH, "//h1[contains(@class, 'document-title')]//span"
            )
            title = title_element.text.strip()
            return title
        except Exception as e:
            print("Error extracting title:", e)
```

INSIDE WEB_SCRAPING.PY

Some functions require button clicks while others requires data processing.

This function has to navigate to the button first, then it has to do string operations to retrieve the necessary data.

SCRAPING

```python
def __find_authors(self, browser):
    try:
        # Find and click the authors button
        authors_button = WebDriverWait(browser, self.delay).until(
            EC.element_to_be_clickable((By.ID, "authors"))
        )
        browser.execute_script("arguments[0].scrollIntoView(true);", authors_button)
        authors_button.click()
        browser.implicitly_wait(self.delay)

        # Scrape authors
        author_elements = browser.find_elements(
            By.XPATH, "//div[contains(@class, 'author-card')]//a/spa
        )
        author_names = [
            author.text for author in author_elements if author.text
        ]
        return author_names
    except Exception as e:
        print("Error extracting authors:", e)
```

```python
def __find_institution_and_location(self, browser):
    try:
        # Find the institution and add it to a list
        institution_elements = browser.find_elements(
            By.XPATH, "//div[contains(@class, 'author-card')]//div[not(@class)]"
        )
        seen_institutions = set()
        institution_texts = [
            institution.text.strip()
            for i, institution in enumerate(institution_elements)
            if institution.text.strip()
            and i % 2 != 0
            and institution.text.strip() not in seen_institutions
            and not seen_institutions.add(institution.text.strip())
        ]

        # Split the text of each institution information to country, city, and institution
        institution, city, country = zip(
            *[
                tuple(map(str.strip, object.split(",")[-3:]))
                for object in institution_texts
            ]
        )
        country = list(set(country))
        city = list(set(city))
        institution = list(set(institution))

        return country, city, institution
```

INSIDE WEB_SCRAPING.PY

```python
def pack_to_json(
    self, title, date, authors, country, city, institution, keywords, filename
):
    data = {
        "Title": title,
        "Authors": authors,
        "Date": date,
        "Institution": institution,
        "City": city,
        "Country": country,
        "Keywords": keywords,
    }

    # Write the data to the JSON file
    with open(filename, "w", encoding="utf-8") as json_file:
        json.dump(data, json_file, ensure_ascii=False, indent=4)

    print(f"Data saved to {filename}")
```

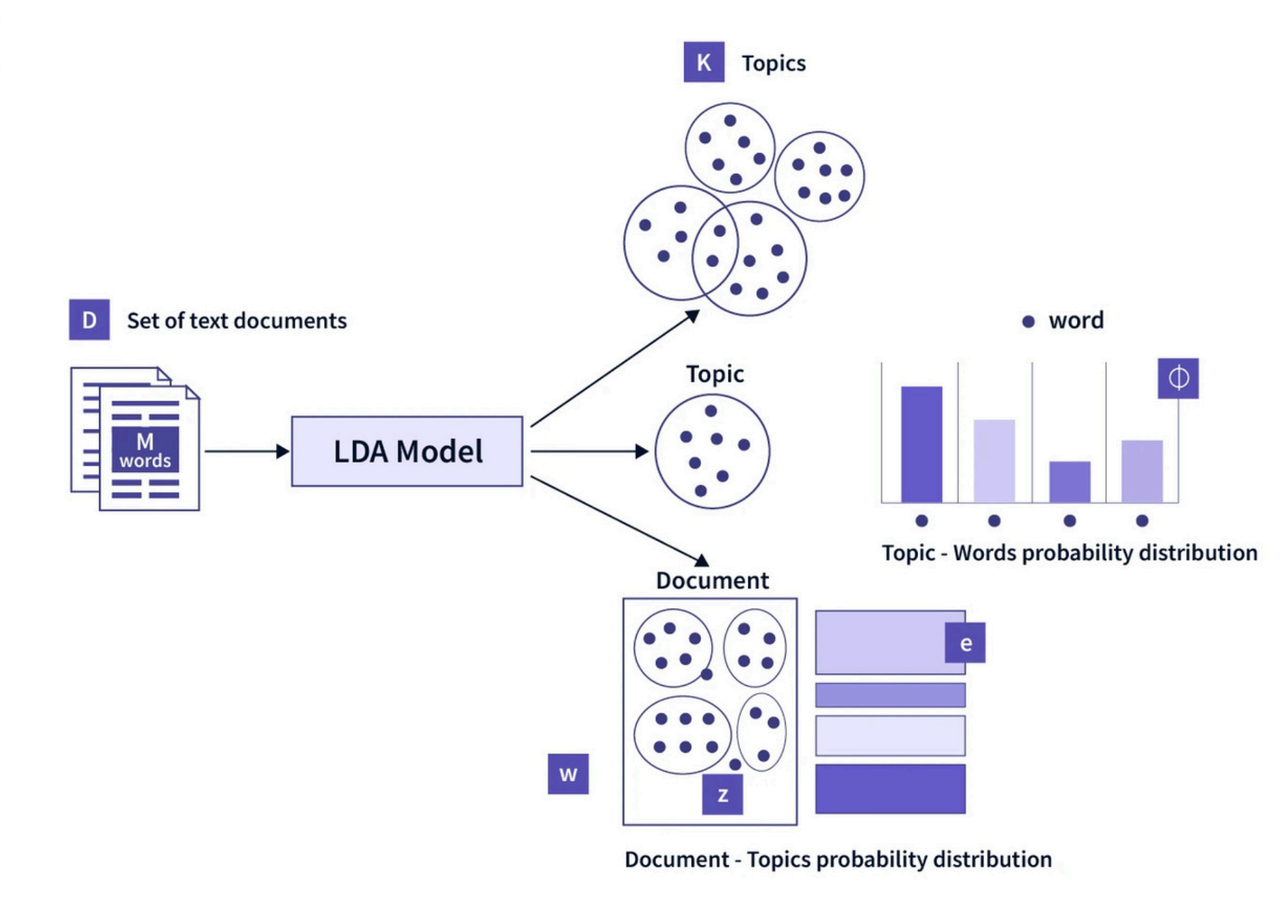
After we get the required information from the site, we store it into a json file so that we don’t lose it after stopping the program.

STORING DATA

MODEL TRAINING

- Latent Dirichlet Allocation (LDA) + GridSearchCV
- K-Mean Clustering

LATENT DIRICHLET ALLOCATION (LDA)



- Identifies latent topics in a corpus by clustering words frequently used together.
- Each topic is a distribution of keywords, and each document has a proportion of topics.

LATENT DIRICHLET ALLOCATION (LDA)

Extract Keywords

```
documents = df['Keywords'].dropna().drop_duplicates().to_list()
vectorizer = CountVectorizer(stop_words='english')
doc_term_matrix = vectorizer.fit_transform(documents)
```

✓ 0.3s

```
["['Random Sampling', 'Radar Sensor', 'Range Compression', 'Radar signal proces
 "['Variable Gain', 'Matching Circuit', 'Intermediate Frequency Signal', 'Outpu
 "['Localization Error', 'Wireless communication', 'Semidefinite Programming',
 "['Frame Structure', 'Multiple-input Multiple-output Systems', 'Converter Modu
 "['Video Games', 'Discrete Grid', 'Deep Learning', '3D Reconstruction', 'Balan
 "['IEC', 'Task Execution', 'sequential task execution', 'Human Subjects', 'Evo
 "['Drag Force', 'quadrotor', 'Neural Network', 'Reinforcement learning', 'Unde
 "['distortion correction', 'Kalman Gain', 'Laser radar', 'Scan Data', 'Pitch A
 "['Input Image', 'Lighting', 'Image color analysis', 'Color Distortion', 'Conv
 "['Ray Tracing', 'Buildings', 'Convolutional Layers', 'Deep Learning', 'Learni
 "['Stability Control', 'X-in-the-loop', 'Electronic Stability Control', 'Test
 "['Two-stage Stochastic Model', 'Carbon emissions trading (CET)', 'Simulation'
 "['Characteristic Signals', 'Time Domain', 'Production', 'Visualization', 'Too
 "['Service robots', 'New Variables', 'Ability Of The Algorithm', 'Root Mean Sc
 "['Low voltage', 'Multifunctional Circuits', 'Signal processing', 'Differentia
 "['Compound Annual Growth Rate', 'Developed Countries', 'Automation', 'Average
 "['Current measurement', 'Genetic algorithm', 'Waveform', 'Amplitude Frequency
```

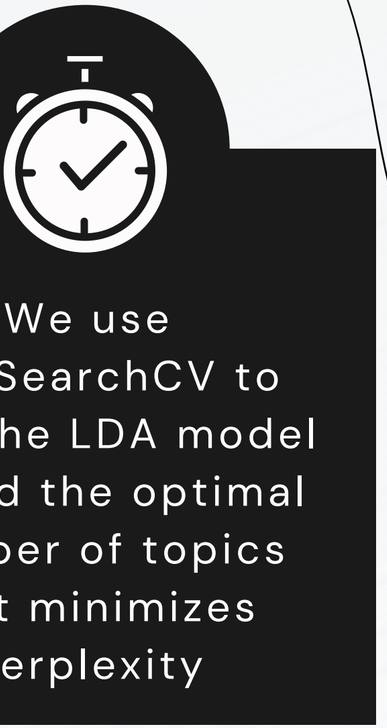
Extract keywords and transform them into a document-term matrix (counts how many times each word appears in each document)

list of keywords

LATENT DIRICHLET ALLOCATION (LDA)

GridSearchCV

```
def perplexity_scorer(estimator, X):
    return -estimator.perplexity(X)

lda = LatentDirichletAllocation(random_state=42, max_iter=50, learning_method='online')

param_grid = {'n_components': [10, 20, 50]}


grid_search = GridSearchCV(lda, param_grid, cv=3, scoring=perplexity_scorer, verbose=2, n_jobs=-1)
grid_search.fit(doc_term_matrix)

best_model = grid_search.best_estimator_
best_n_topics = grid_search.best_params_['n_components']
print(f"Best number of topics: {best_n_topics}")
```



We use GridSearchCV to tune the LDA model to find the optimal number of topics that minimizes perplexity

- we test for
  - 10, 20, 50 topics
  - 3 cross folds

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
Best number of topics: 10
Model and best number of topics saved.
```

+ Code + Markdown

Result is 10 topics is the best

LATENT DIRICHLET ALLOCATION (LDA)

Fit LDA

```
# Now use the optimal number of topics in your LDA model
lda = LatentDirichletAllocation(n_components=optimal_topics, random_state=42)
lda.fit(doc_term_matrix)
```

- From the model we generate a list of topic names by using lda_model.components._

```
# Generate topic names
topic_names = [
    " ".join([vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-5:][::-1]])
    for topic in lda_model.components_
]
```

fits the LDA with the best amount of topics that we got from the previous slides and find the topics from the keywords

LATENT DIRICHLET ALLOCATION (LDA)

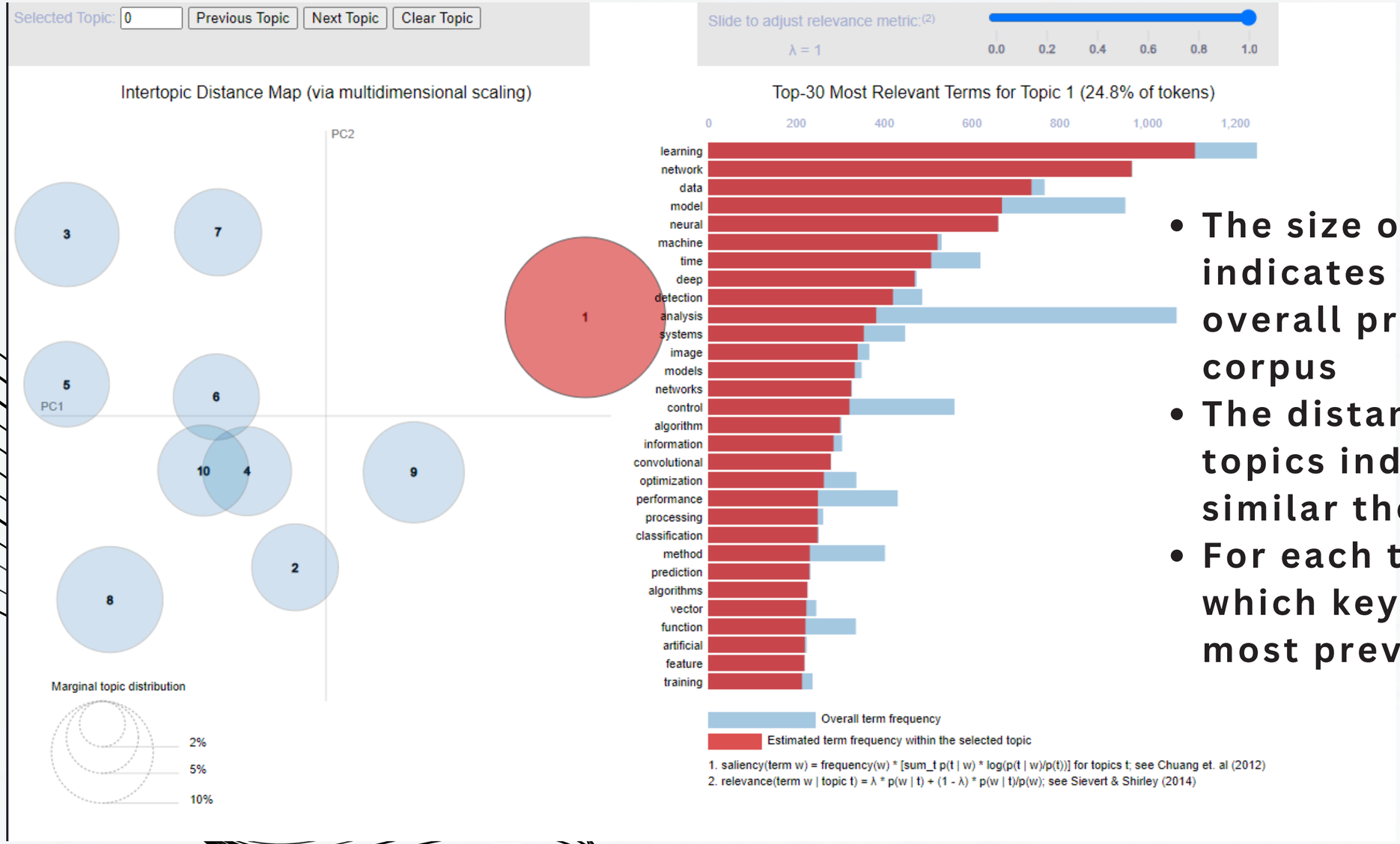
Map each topic we got into a proper subject area name

```
Topic Names:
Topic 1: learning network data model neural
Topic 2: cells frequency antenna communication array
Topic 3: cancer health stress cell thailand
Topic 4: covid 19 analysis optical acid
Topic 5: hiv thailand bone cells poly
Topic 6: control species activity antioxidant thailand
Topic 7: vaccine learning drug thai molecular
Topic 8: carbon oil acid properties kidney
Topic 9: energy power voltage analysis electric
Topic 10: thailand social business based technology
```

```
new_column_names = {
    'learning network data model neural': 'Artificial Intelligence',
    'cells frequency antenna communication array': 'Telecommunication Engineering',
    'cancer health stress cell thailand': 'Medical Science',
    'covid 19 analysis optical acid': 'Disease Analysis',
    'hiv thailand bone cells poly': 'Biomedical Research',
    'control species activity antioxidant thailand': 'Environmental Science',
    'vaccine learning drug thai molecular': 'Pharmaceutical Science',
    'carbon oil acid properties kidney': 'Chemical Science',
    'energy power voltage analysis electric': 'Energy and Power Systems',
    'thailand social business based technology': 'Social and Business Studies',
}
```

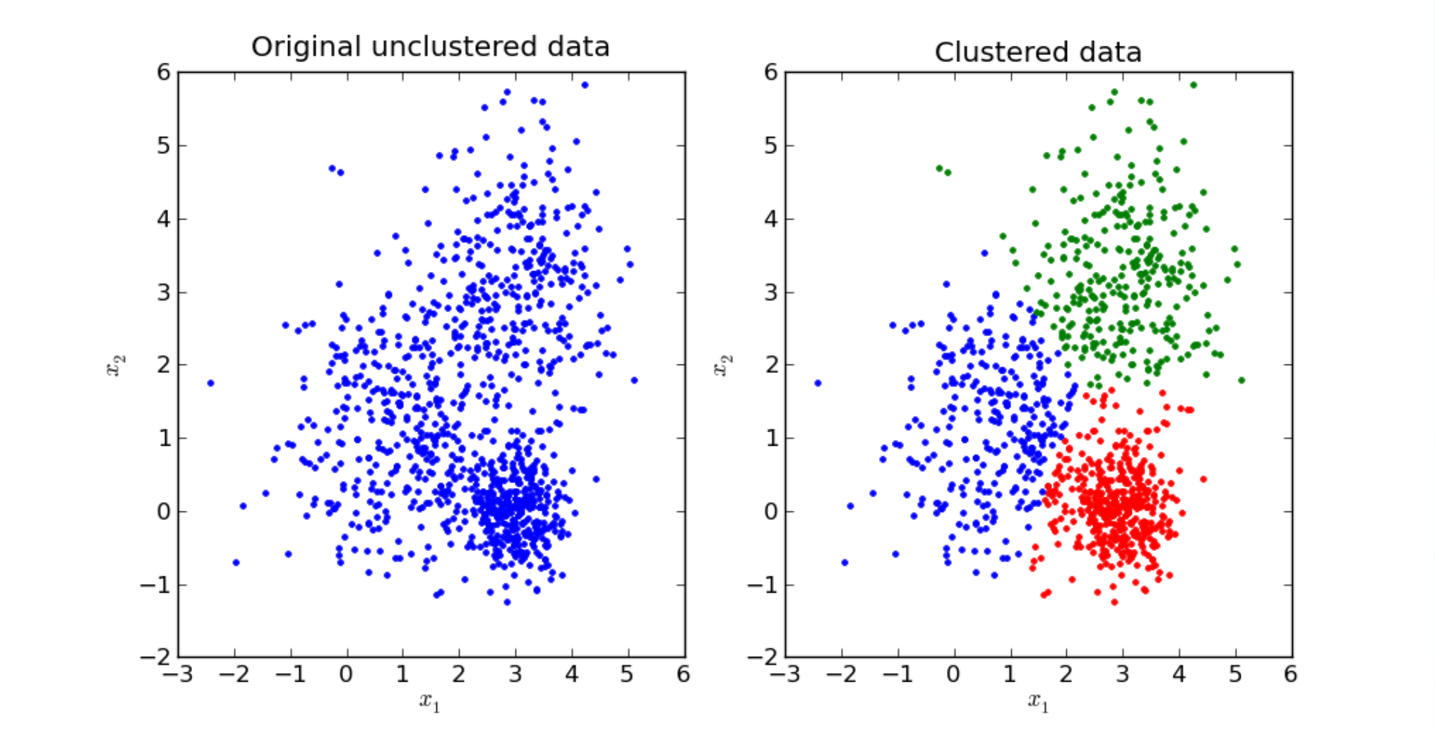
LATENT DIRICHLET ALLOCATION (LDA)

Visualization (pyLDAvis)



- The size of the circle indicates the topic's overall prevalence in the corpus
- The distance between topics indicates how similar they are
- For each topic, we can see which keywords are the most prevalent

K-MEANS CLUSTERING



- **Combine LDA with K-Means Clustering** to analyze the underlying structure in a corpus documents based on their topic distributions
- Identify the dominant institutions for each topic cluster

K-MEANS CLUSTERING

Clustering with K-Means

```
num_clusters = lda.n_components
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
clusters = kmeans.fit_predict(doc_topic_matrix)
filtered_df['Cluster'] = clusters
```

create a cluster for each topic and use KMeans model fitted on the document matrix

We identify the institution specializing in each cluster and in the DataFrame, add the mean topic distributions and the cluster the institution is in

```
cluster_topic_means = pd.DataFrame(doc_topic_matrix).groupby(clusters).mean()
cluster_topic_means.columns = [f"Topic {i}" for i in range(1, num_clusters + 1)]

cluster_specialization = filtered_df.groupby('Cluster')['Institution'].apply(
    lambda group: group.value_counts().idxmax())

cluster_names = cluster_topic_means.idxmax(axis=1).apply(lambda x: topic_names[int(x.split()[-1]) - 1])


cluster_specialization = filtered_df.groupby('Cluster')['Institution'].apply(
    lambda group: group.value_counts().idxmax()
)

filtered_df['Cluster Keywords'] = filtered_df['Cluster'].map(cluster_names)
cluster_analysis = pd.concat([cluster_topic_means, cluster_specialization.rename('Top Institution')], axis=1)
```

Diversity of each subject area for that institute

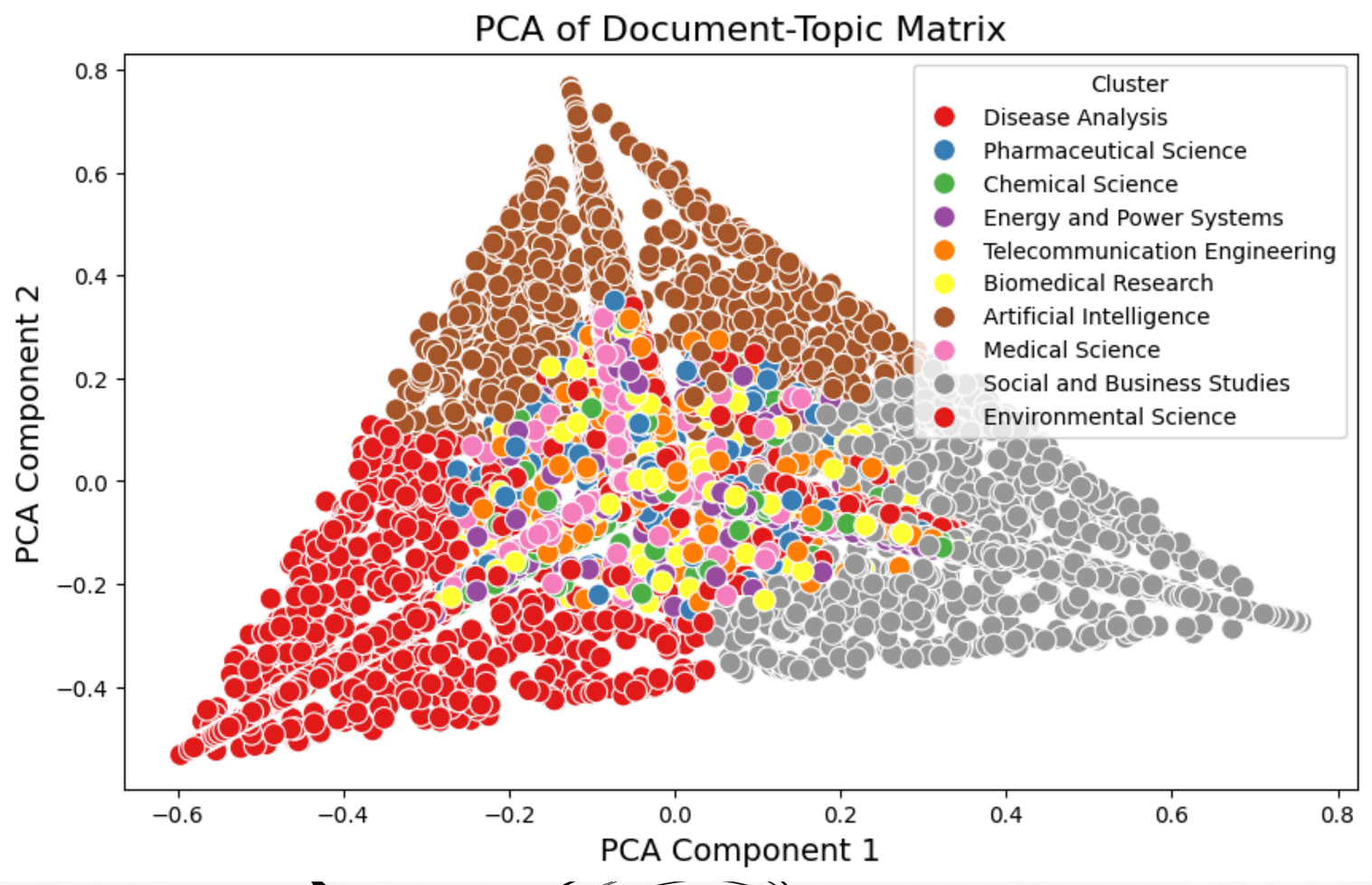
The cluster the institute is in aka the subject area they specialize in

	Institution	Country	Artificial Intelligence	Telecommunication Engineering	Medical Science	Disease Analysis	Biomedical Research	Environmental Science	Pharmaceutical Science	Chemical Science	Energy and Power Systems	Social and Business Studies	Cluster	Cluster Keywords	Subject Area
0	Ruhr University Bochum	Germany	0.833055	0.103295	0.001515	0.001515	0.001515	0.053044	0.001515	0.001515	0.001515	0.001515	3	learning network data model neural	Artificial Intelligence
1	Technische Universität Dresden	Germany	0.258691	0.002001	0.002000	0.002000	0.002000	0.422623	0.002000	0.002000	0.304685	0.002000	6	control species activity antioxidant thailand	Environmental Science
2	University of Duisburg-Essen	Germany	0.734131	0.160623	0.001031	0.001031	0.001031	0.001031	0.001031	0.001031	0.001031	0.098028	3	learning network data model neural	Artificial Intelligence
3	Sakura-shi	Japan	0.451511	0.484551	0.001111	0.001111	0.001111	0.001111	0.001111	0.001111	0.056159	0.001111	7	cells frequency antenna communication array	Telecommunication Engineering
4	University of Surrey	United Kingdom	0.817539	0.038230	0.001539	0.133461	0.001539	0.001539	0.001539	0.001539	0.001539	0.001539	3	learning network data model neural	Artificial Intelligence
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17107	Sezione di Firenze	France	0.007144	0.007145	0.007145	0.007145	0.007146	0.007144	0.007144	0.007144	0.007144	0.935698	1	thailand social business based technology	Social and Business Studies
17108	Recinto Universitario de Mayagüez	France	0.008334	0.008334	0.008337	0.008334	0.008334	0.008334	0.008334	0.519437	0.008334	0.413889	9	carbon oil acid properties kidney	Chemical Science

K-MEANS CLUSTERING

Visualization (PCA)

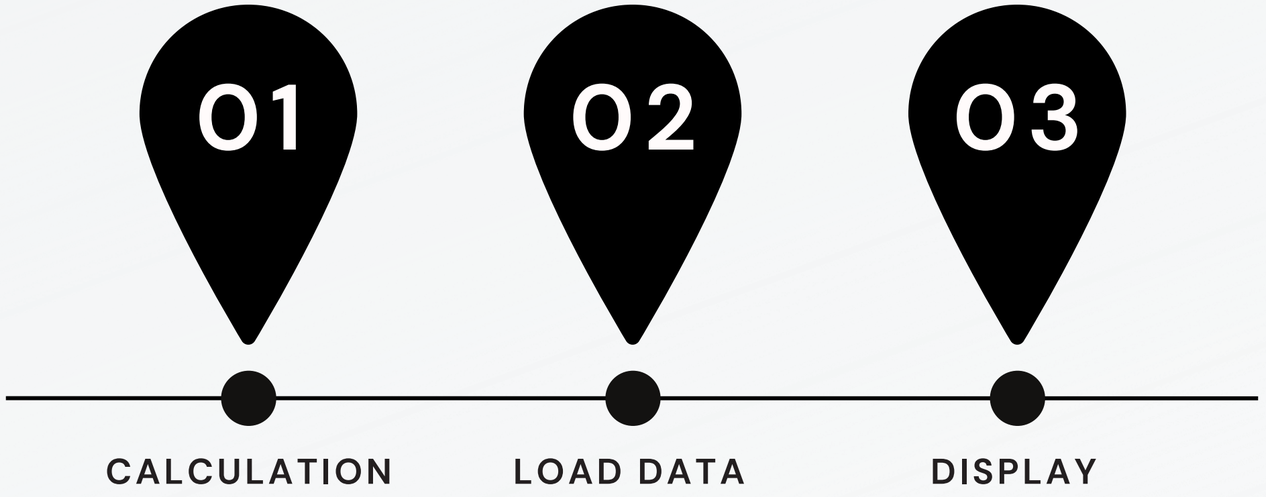
fun fact: we can see here that disease analysis, social and business studies, and environment science are the most dominant topics



- Each dot represents different institutions
- The colors represents different subject area (cluster)

- **Institution within the same cluster are similar in terms of topic they focus on**

DATA VISUALISATION



DATA VISUALISATION

CALCULATION

We start off by calculating some data first so that we dont have to waste time recalculating everything again.

```python
def get_country_coordinates(country_name):
    try:
        geolocator = Nominatim(user_agent="country_geocoder")
        location = geolocator.geocode(country_name)
        if location:
            return location.latitude, location.longitude
        else:
            return None, None
    except Exception as e:
        return None, None


def calculate_map(df_dataset, output_filename="country_frequency.csv"):
    df = df_dataset
    df["Country"] = df["Country"].apply(ast.literal_eval)
    country_list = [country for sublist in df["Country"] for country in sublist]
    country_freq = pd.Series(country_list).value_counts().reset_index()
    country_freq.columns = ["country", "frequency"]
    country_freq["coordinates"] = country_freq["country"].apply(get_country_coordinates)
    country_freq = country_freq[
        country_freq["coordinates"].apply(lambda x: x != (None, None))
    ]
    country_freq["latitude"] = country_freq["coordinates"].apply(lambda x: x[0])
    country_freq["longitude"] = country_freq["coordinates"].apply(lambda x: x[1])

    # Save to CSV
    country_freq.to_csv(output_filename, index=False)

    return country_freq
```

DATA VISUALISATION

```python
@st.cache_data
def load_dataset(file_path):
    try:
        data = pd.read_csv(file_path, on_bad_lines="warn")
        return data
    except Exception as e:
        st.error(f"Failed to load data: {e}")
        return pd.DataFrame()


@st.cache_data
def load_cluster_data(file_path):
    try:
        data = pd.read_csv(file_path, on_bad_lines="warn")
        return data
    except Exception as e:
        st.error(f"Failed to load cluster data: {e}")
        return pd.DataFrame()


@st.cache_data
def load_map_data(file_path):
    try:
        data = pd.read_csv(file_path, on_bad_lines="warn")
        return data
    except Exception as e:
        st.error(f"Failed to load map data: {e}")
        return pd.DataFrame()
```

LOAD DATA

We then load the data into StreamLit, making sure to cache it so that we don’t have to reload them.

DATA VISUALISATION

DISPLAY

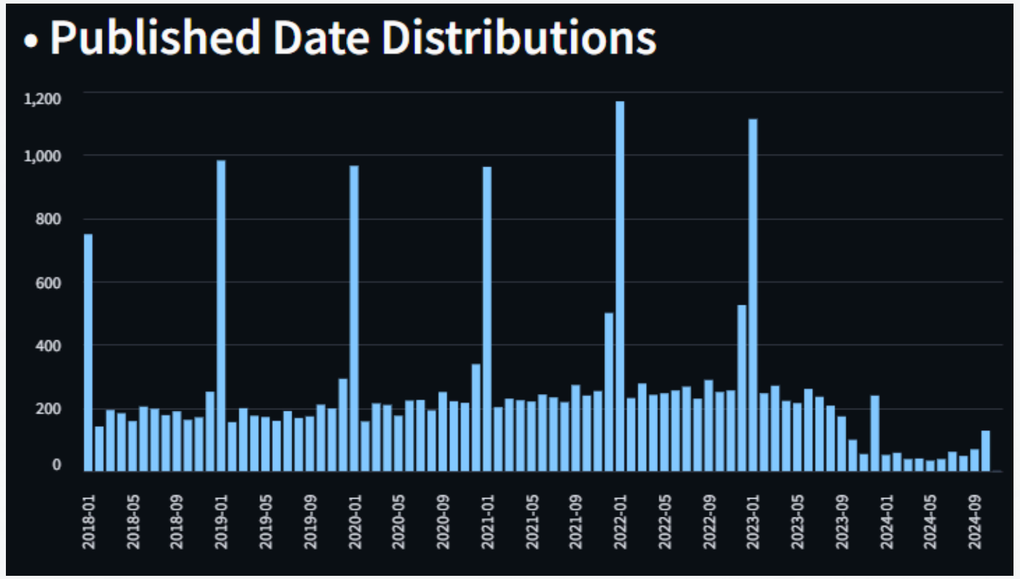
We can display the data in a way that is interesting and offers a better understanding of the dataset and training model.

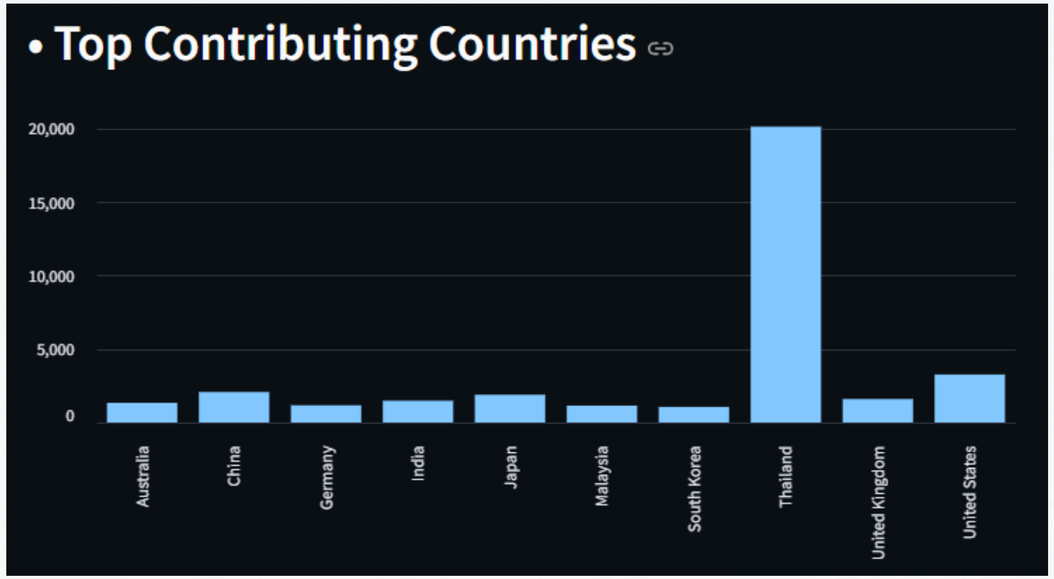
```python
def show_title():
    st.title(
        "How do the affiliations of researchers influence the diversity of engineering research topics?"
    )


def show_overview(data):
    st.header("• Exploring Our Data set")
    if st.checkbox("Data Preview"):
        st.subheader("Examples of the Data Set")
        st.write(data.head())
        st.subheader("Data Set Statistics")
        st.write(data.describe().head(2))
```

```python
def show_map(df_dataset, df_map):
    df = df_dataset
    country_freq = df_map
    st.header("• Where Does Our Data Come From?")
    st.map(country_freq)
```

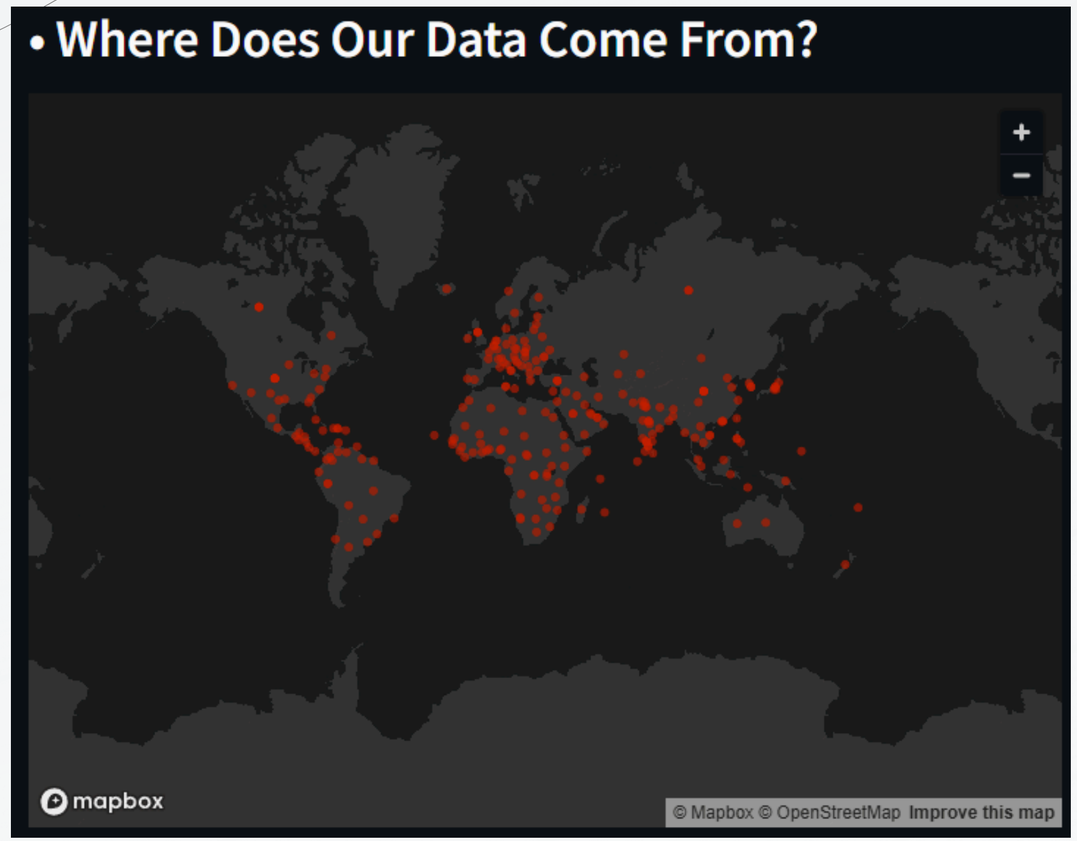
```python
def show_date(dataset):
    st.header("• Published Date Distributions")
    dataset["Date"] = pd.to_datetime(dataset["Date"], errors="coerce")
    dataset = dataset.dropna(subset=["Date"])
    dataset["Year-Month"] = dataset["Date"].dt.to_period("M").astype(str)
    date_data = dataset.groupby("Year-Month")["Title"].count()
    st.bar_chart(date_data)
```

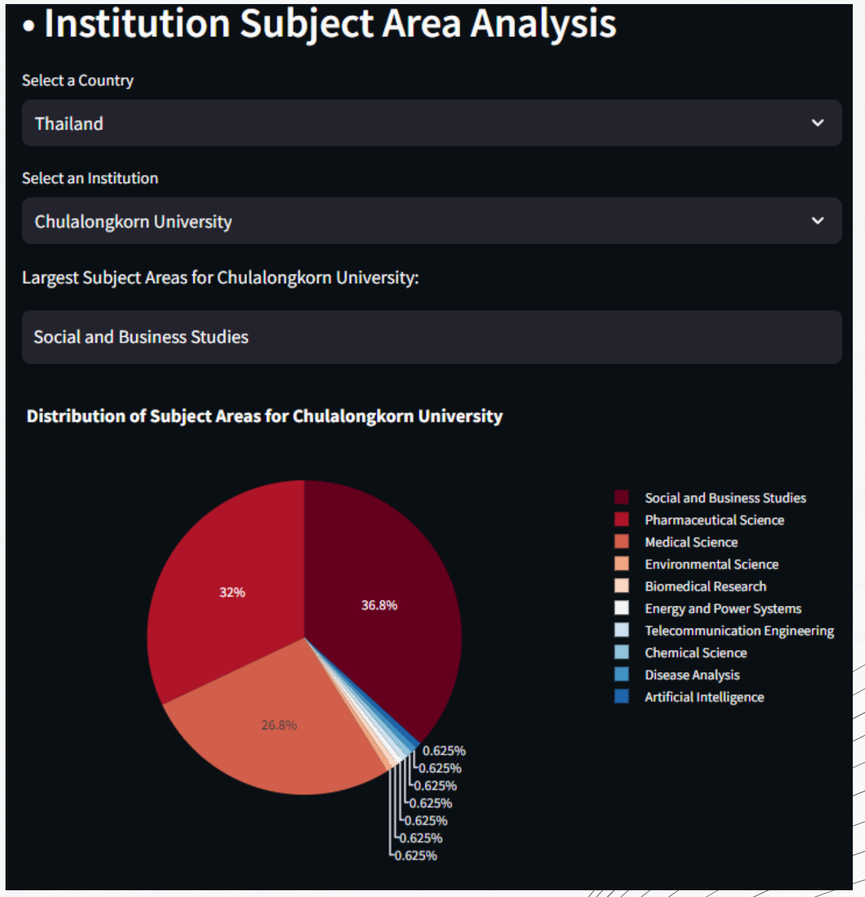
DATA VISUALISATION



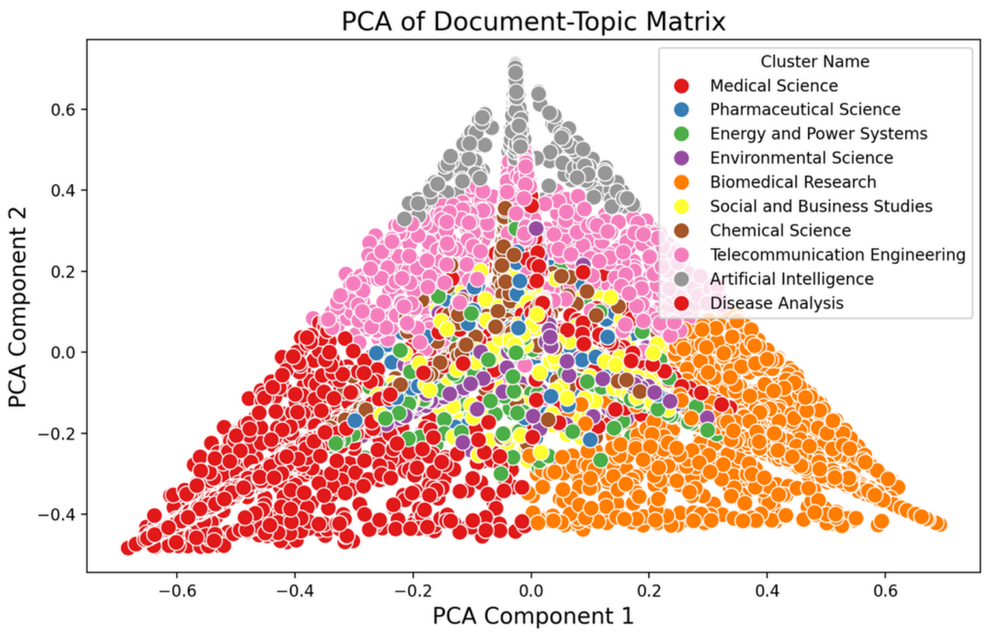


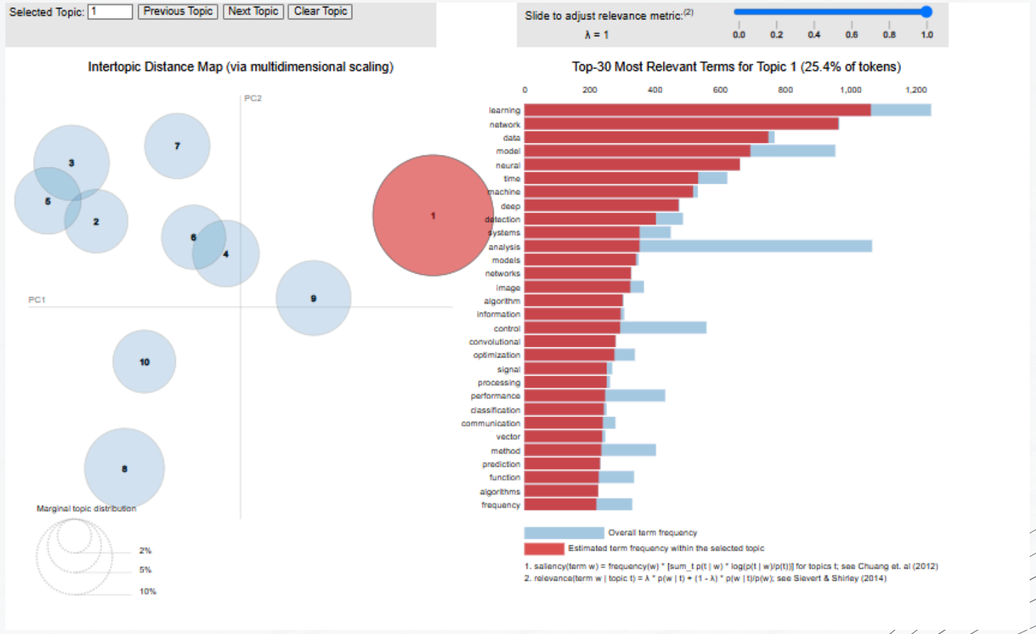
DATA VISUALISATION





DATA VISUALISATION





THANKS FOR WATCHING

GITHUB LINK:

HTTPS://GITHUB.COM/TERIYAKITHAMES/DATA-SCIENCE-PROJECT