

# **RESEARCH SUBJECT**

# **PREDICTION PROJECT**

# WHAT WE DO



## Input

organizations, classifications,  
affiliations, auth-keywords,  
reference, or publish name

## Subject Prediction

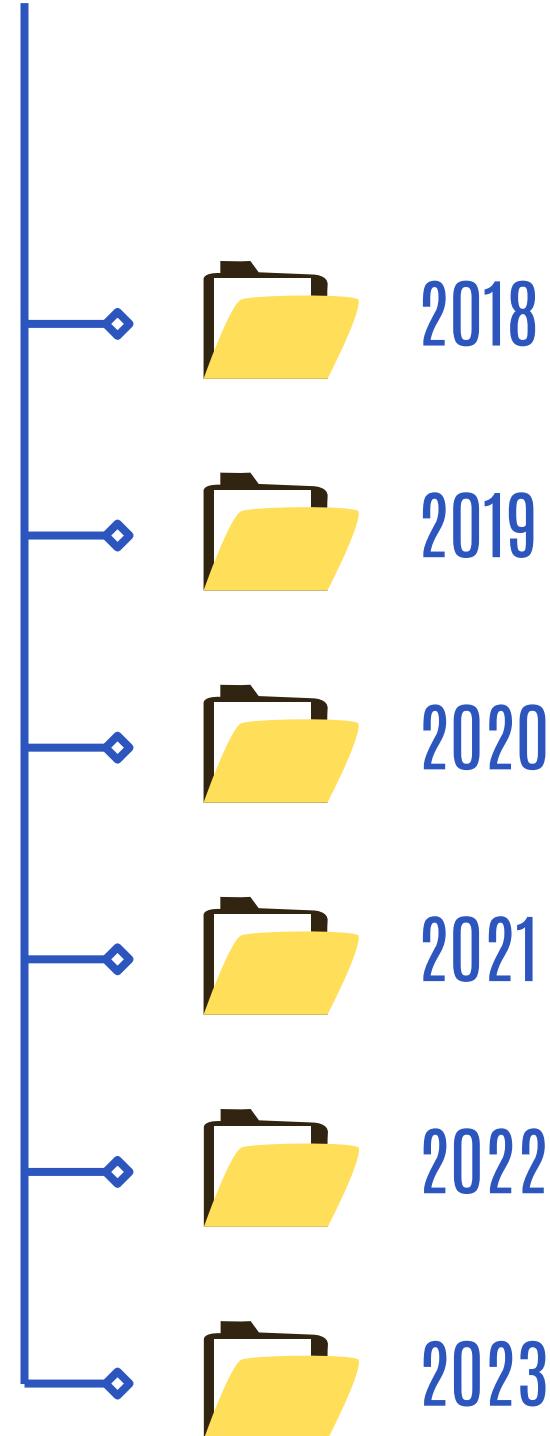
using our trained model

## Return the most compatible subject

along with total views and  
rating for the book in the same  
subject

# ABOUT OUR DATA USED

# RAW-DATA



SOURCE DATA :

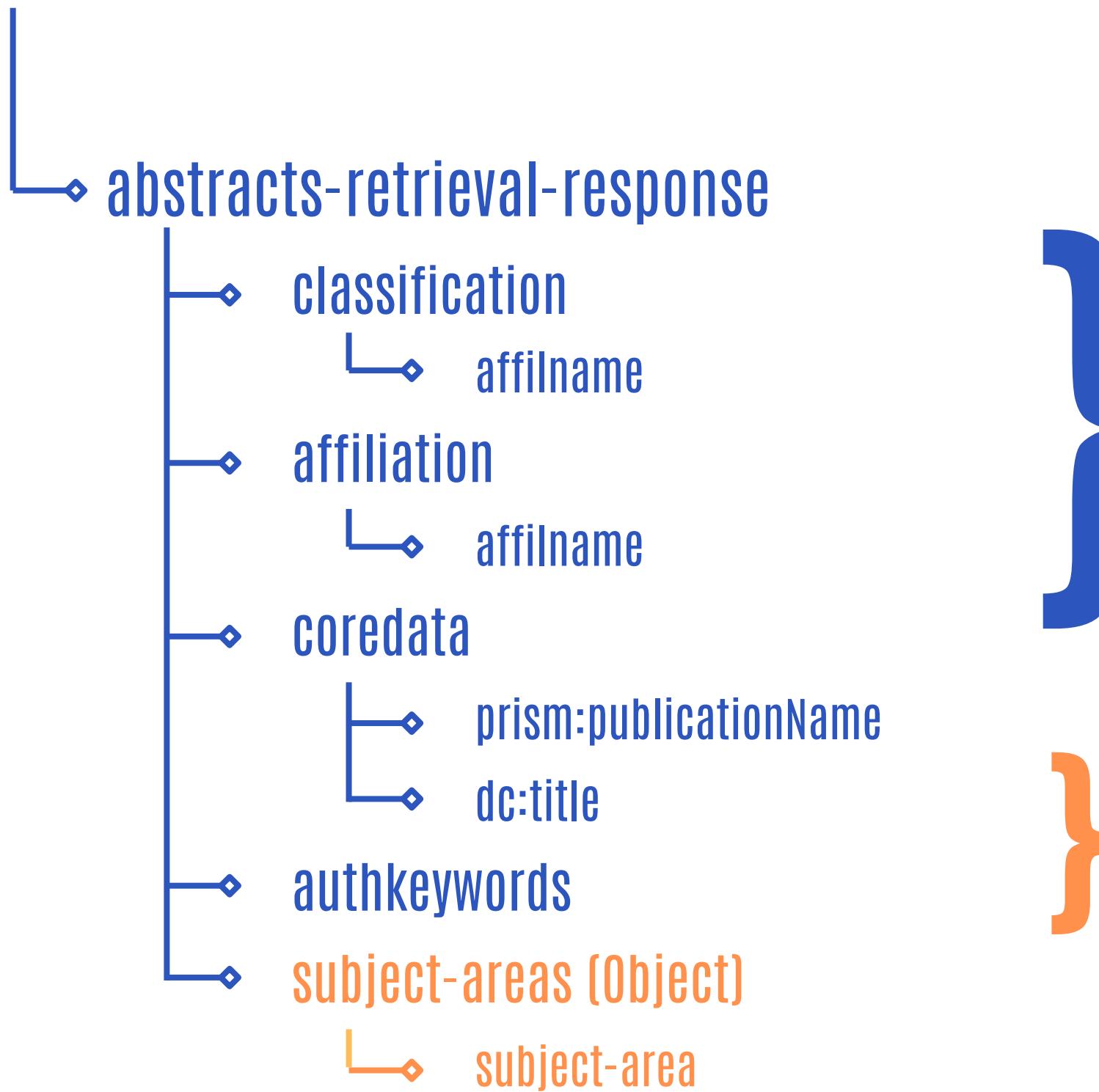


**20,216  
FILES**

# DATA PREPARATION

ABOUT  
OUR

# FILE SCHEMA



} INPUT VARIABLES  
}

} OUTPUT VARIABLES

# DATA-PREPARATION

## GET FEATURES

- File
- Organizations
- Classifications
- Affiliations
- Auth-keywords
- Authors
- Reference
- Title
- Publish\_name
- Subjects

```
def extract_features(json_file):
    with open(json_file, 'r', encoding='utf-8') as f:
        data = json.load(f)

    abs_resp = data.get('abstracts-retrieval-response', {})

    # Special handling for organizations
    def get_organizations(data):
        orgs = []
        author_groups = data.get('item', {}).get('bibreCORD', {}).get('head', {}).get('author-group', [])

        if isinstance(author_groups, dict):
            author_groups = [author_groups]

        for group in author_groups:
            if isinstance(group, dict):
                affiliation = group.get('affiliation', {})
                if isinstance(affiliation, dict):
                    org = affiliation.get('organization', [])
                    if isinstance(org, dict):
                        org = [org]
                    if isinstance(org, list):
                        for o in org:
                            if isinstance(o, dict):
                                orgs.append(o.get('$', None))
                            else:
                                orgs.append(o)

        return orgs if orgs else [None]

    features = {
        'organizations': {
            'custom': get_organizations
        },
        'classifications': {
            'path': ['item', 'bibreCORD', 'head', 'enhancement',
                     'classificationgroup', 'classifications'],
            'key': '#type'
        },
        'affiliations': {
            'path': ['affiliation'],
            'key': 'affilname'
        },
        'auth-keywords': {
            'path': ['authkeywords', 'author-keyword'],
            'key': '$'
        },
        'subjects': {
            'path': ['subject-areas', 'subject-area'],
            'key': '$'
        },
        'authors': {
            'path': ['authors', 'author'],
            'key': 'ce: indexed-name'
        }
    }

    record = {'file': Path(json_file).name}

    for feature_name, config in features.items():
        if 'custom' in config:
            # Use custom extraction function for organizations
            record[feature_name] = config['custom'](abs_resp)
        elif 'subpath' in config:
            values = []
            main_array = get_array_or_dict_values(abs_resp, config['path'], None)
            for item in main_array:
                if isinstance(item, dict):
                    sub_values = get_array_or_dict_values(item, config['subpath'], config['key'])
                    values.extend(sub_values)
            record[feature_name] = values
        else:
            record[feature_name] = get_array_or_dict_values(abs_resp, config['path'], config['key'])

    return record
```

```
def extract_features(json_file):
    with open(json_file, 'r', encoding='utf-8') as f:
        data = json.load(f)

    abs_resp = data.get('abstracts-retrieval-response', {})

    # Special handling for organizations
    def get_organizations(data):
        orgs = []
        author_groups = data.get('item', {}).get('bibreCORD', {}).get('head', {}).get('author-group', [])

        if isinstance(author_groups, dict):
            author_groups = [author_groups]

        for group in author_groups:
            if isinstance(group, dict):
                affiliation = group.get('affiliation', {})
                if isinstance(affiliation, dict):
                    org = affiliation.get('organization', [])
                    if isinstance(org, dict):
                        org = [org]
                    if isinstance(org, list):
                        for o in org:
                            if isinstance(o, dict):
                                orgs.append(o.get('$', None))
                            else:
                                orgs.append(o)

        return orgs if orgs else [None]

    features = {
        'organizations': {
            'custom': get_organizations
        },
        'classifications': {
            'path': ['item', 'bibreCORD', 'head', 'enhancement',
                     'classificationgroup', 'classifications'],
            'key': '#type'
        },
        'affiliations': {
            'path': ['affiliation'],
            'key': 'affilname'
        },
        'auth-keywords': {
            'path': ['authkeywords', 'author-keyword'],
            'key': '$'
        },
        'subjects': {
            'path': ['subject-areas', 'subject-area'],
            'key': '$'
        },
        'authors': {
            'path': ['authors', 'author'],
            'key': 'ce: indexed-name'
        }
    }

    record = {'file': Path(json_file).name}

    for feature_name, config in features.items():
        if 'custom' in config:
            # Use custom extraction function for organizations
            record[feature_name] = config['custom'](abs_resp)
        elif 'subpath' in config:
            values = []
            main_array = get_array_or_dict_values(abs_resp, config['path'], None)
            for item in main_array:
                if isinstance(item, dict):
                    sub_values = get_array_or_dict_values(item, config['subpath'], config['key'])
                    values.extend(sub_values)
            record[feature_name] = values
        else:
            record[feature_name] = get_array_or_dict_values(abs_resp, config['path'], config['key'])

    return record
```

```
def get_array_or_dict_values(data, path, key):
    """Extract values from deeply nested arrays/dictionaries."""
    try:
        value = data
        for p in path:
            if isinstance(value, dict):
                value = value.get(p, None)
            elif isinstance(value, list):
                # Handle list of dicts or list of lists
                temp_values = []
                for item in value:
                    if isinstance(item, dict):
                        item_value = item.get(p, None)
                        if isinstance(item_value, list):
                            temp_values.extend(item_value)
                        elif item_value is not None:
                            temp_values.append(item_value)
                    elif isinstance(item, list):
                        temp_values.extend(item)
                value = temp_values if temp_values else None
            else:
                return [None]

        # Handle final value
        if value is None:
            return [None]
        elif isinstance(value, dict):
            # Handle dictionary case
            if key in value:
                return [value[key]]
            elif '$' in value: # Special case for $ key
                return [value['$']]
            return [None]
        elif isinstance(value, list):
            # Handle list case
            results = []
            for item in value:
                if isinstance(item, dict):
                    results.append(item[key])
                elif '$' in item: # Special case for $ key
                    results.append(item['$'])
                else:
                    results.append(None)
            return results
        else:
            return [value]
    except Exception as e:
        print(f"Error extracting from path {path}: {e}")
        return [None]
```

# DATA-PREPARATION

See information

```
df_data_file = pd.read_csv("data/features_summation.csv", encoding='utf-8')
df_data_file.info()
df_data_file.describe(include='all')
df_data_file.shape
df_data_file.head(5)
```

Drop column and null

```
df_data_file.drop(columns=['file', 'authors', 'reference','title'], inplace=True)
df_data_file_null_out = df_data_file_null_out.applymap(lambda x: None if x == '[None]' or x == '[]' or x == 'None' else x)
df_data_file_null_out.dropna(inplace=True)
df_data_file_null_out.drop_duplicates(inplace=True)
```

# DATA-PREPARATION

Drop column value that has count < 1 % and > 99 %

```
# Calculate the 1st and 99th percentiles
lower_percentile = df_data_file_cutoff['auth-keywords'].value_counts().quantile(0.01)
upper_percentile = df_data_file_cutoff['auth-keywords'].value_counts().quantile(0.99)

# Filter the DataFrame to keep only the rows with 'auth-keywords' within the desired range
df_data_file_cutoff = df_data_file_cutoff[
    df_data_file_cutoff['auth-keywords'].map(df_data_file_cutoff['auth-keywords'].value_counts()).between(lower_percentile, upper_percentile)
]

# Display the shape of the new DataFrame
df_data_file_cutoff.shape
```

# DATA-PREPARATION

## DATA PREPARATION

### ↳ Preparing each input variable

- ❖ Organizations
- ❖ Classifications
- ❖ Affiliations
- ❖ Auth-keywords
- ❖ Publish\_name
- ❖ Subjects

```
df_data_file_subject = pd.read_csv('data_pandas/1_features_drop_null.csv', encoding='utf-8')
df_data_file_subject.shape
df_data_file_subject.describe(include='all')

# Ensure the subjects column contains valid string representations of lists
def safe_literal_eval(val):
    try:
        return ast.literal_eval(val)
    except (ValueError, SyntaxError):
        return []

# Split the subjects column into multiple rows
df_data_file_subject['subjects'] = df_data_file_subject['subjects'].apply(safe_literal_eval)

df_data_file_subject = df_data_file_subject.explode('subjects').reset_index(drop=True)

# Clean subject values
df_data_file_subject['subject'] = (
    df_data_file_subject['subjects']
    .str.replace(r'[\n\r\v]', '', regex=True) # Remove unwanted characters
    .str.strip() # Trim whitespace
    .str.replace(r'\(.?\)', '', regex=True) # Remove parentheses and their contents
    .str.lower() # Convert to lowercase
)

def is_valid_utf8(text):
    try:
        text.encode('utf-8')
        return True
    except UnicodeEncodeError:
        return False

# Filter rows with only valid UTF-8 characters
df_data_file_subject['subject'] = df_data_file_subject['subject'].apply(lambda x: x if is_valid_utf8(x) else np.nan)

# Drop the original subjects column (if required)
df_data_file_subject.drop(columns=['subjects'], inplace=True)

df_data_file_subject.dropna(inplace=True)
df_data_file_subject.drop_duplicates(inplace=True)

import json
with open('clustering_subject.json', encoding='utf-8') as file:
    mapping_subjects = json.load(file)

import nltk
from nltk.stem import PorterStemmer
from fuzzywuzzy import fuzz

subject_cluster_mapping = {cluster_data['Cluster']: set(cluster_data['Subjects']) for cluster_data in mapping_subjects}
stemmer = PorterStemmer()

def map_subjects(subject):
    subject = subject.lower()
    tokens = subject.split()
    stemmed_tokens = [stemmer.stem(token) for token in tokens]

    for cluster, subjects in subject_cluster_mapping.items():
        for cluster_subject in subjects:
            cluster_tokens = cluster_subject.lower().split()
            stemmed_cluster_tokens = [stemmer.stem(token) for token in cluster_tokens]

            # Exact match
            if set(stemmed_tokens) == set(stemmed_cluster_tokens):
                return cluster

            # Fuzzy matching based on token similarity
            fuzzy_ratio = fuzz.token_sort_ratio(subject, cluster_subject)
            if fuzzy_ratio > 80:
                return cluster

    return 'Other'

df_data_file_subject_clustering['subject-cluster'] = df_data_file_subject_clustering['subject'].apply(lambda x: map_subjects(x.strip()))

df_data_file_subject_clustering.drop(columns=['subject'], inplace=True)
```

# ABOUT OUR WEB SCRAPING



# WEB SCRAPING

# GET FEATURES

- Title
  - Author
  - Keyword
  - Reading log count
  - Rating
  - Link

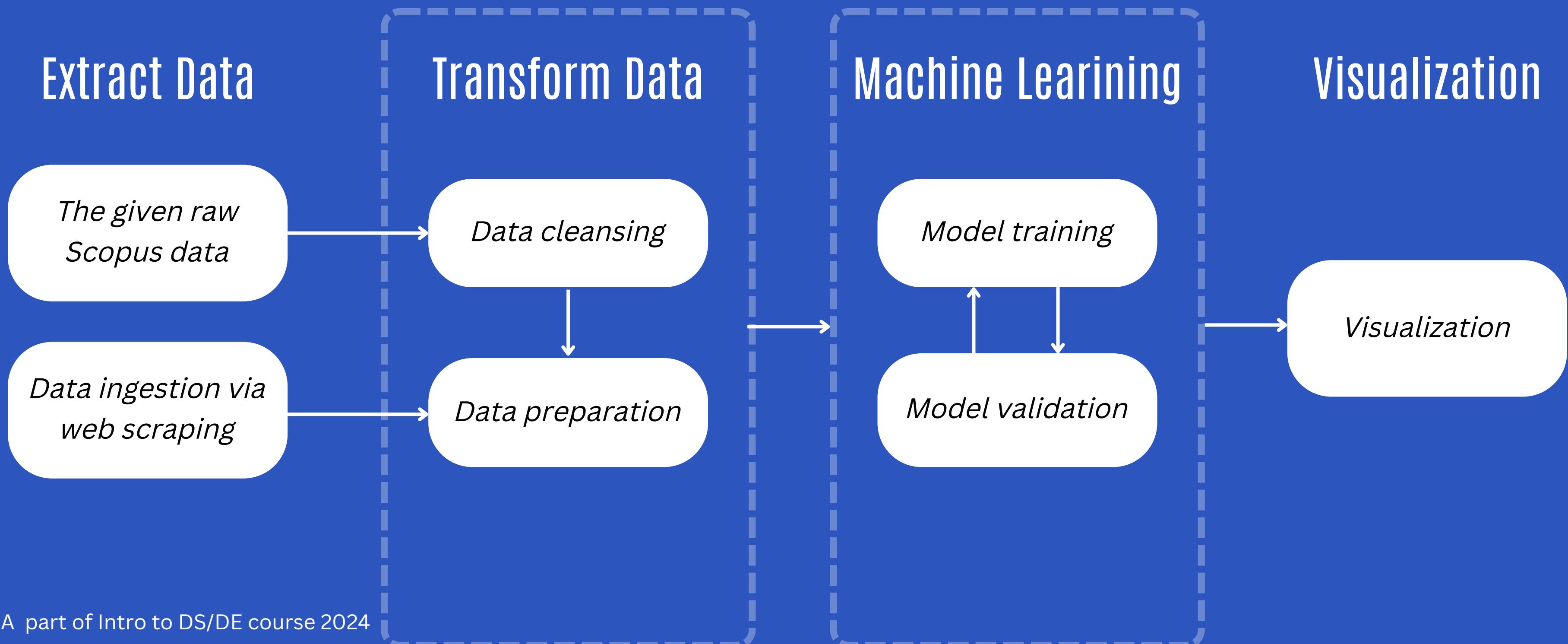
# WEB SCRAPING

	title	author_name	subject_key	readinglog_count	ratings_avg	ratings_count	href
0	NASA/DoD aerospace knowledge diffusion research report	['Thomas E. Pinell']	['aeronautical_engineering', 'aeronautics', 'aerodynamics', 'aircraft', 'aviation', 'flight', 'space', 'spacecraft', 'spaceflight']	168.0	4.000000	5.0	<a href="https://openlibrary.org/works/OL8894965W">https://openlibrary.org/works/OL8894965W</a>
1	Engineering mechanics	['R. C. Hibbeler', 'R.C. Hibbeler', 'S.C. Fan']	['applied_mechanics', 'dynamics', 'dynamik', 'engineering', 'mechanics']	365.0	4.500000	10.0	<a href="https://openlibrary.org/works/OL1865037W">https://openlibrary.org/works/OL1865037W</a>
2	The Soul of a New Machine	['Tracy Kidder']	['architecture_des_ordinateurs', 'computer_engineering', 'computers', 'hardware']	173.0	4.133333	15.0	<a href="https://openlibrary.org/works/OL96218W">https://openlibrary.org/works/OL96218W</a>
3	[William Wheeler Hubbell, authorized to apply for] a bill to prohibit the sale of agricultural machinery and implements in the District of Columbia	['United States. Congress. Senate. Committee on Agriculture, Nutrition, and Forestry']	['administration_of_justice', 'agricultural_machinery', 'agriculture', 'economics', 'machinery']	224.0	3.250000	12.0	<a href="https://openlibrary.org/works/OL7574091W">https://openlibrary.org/works/OL7574091W</a>
4	Pride and Prejudice	['Jane Austen']	['adaptations', 'amour', 'austen_jane_1775-1817', 'classical_literature', 'classic_literature', 'comedy', 'fiction', 'literature']	4737.0	4.131488	289.0	<a href="https://openlibrary.org/works/OL640554W">https://openlibrary.org/works/OL640554W</a>
—	...	—	—	—	—	—	—
1795	On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life	['Charles Darwin']	['biological_evolution', 'classic_literature', 'biology', 'natural_selection']	448.0	4.238095	21.0	<a href="https://openlibrary.org/works/OL515051W">https://openlibrary.org/works/OL515051W</a>
1796	De rerum natura	['Titus Lucretius Carus']	['ancient_philosophy', 'anthologies', 'atomic_theory', 'biology', 'classical_literature', 'latin']	344.0	3.769231	13.0	<a href="https://openlibrary.org/works/OL1548597W">https://openlibrary.org/works/OL1548597W</a>
1797	Biology	['Cecile Starr', 'Christine Evans', 'Lisa Starr']	['biology', 'biology', 'biology_social_aspect', 'classical_literature', 'classical_science', 'classical_studies']	296.0	3.818182	11.0	<a href="https://openlibrary.org/works/OL1975314W">https://openlibrary.org/works/OL1975314W</a>
1798	Biology	['Sylvia S. Mader']	['biology', 'biology', 'biology_textbooks', 'classical_literature', 'classical_science']	136.0	5.000000	1.0	<a href="https://openlibrary.org/works/OL2657947W">https://openlibrary.org/works/OL2657947W</a>
1799	Physics	['Aristotle']	['08.21', '18.40', '3007', 'ancient_science', 'classical_science']	420.0	3.705882	17.0	<a href="https://openlibrary.org/works/OL151632W">https://openlibrary.org/works/OL151632W</a>

1,800 rows × 7 columns

# 1,800 ROWS

# DATA PIPELINE



# DE COMPONENT

Web scraping to recommended books

- Beautifulsoup
-  SpringerLink
-  LIBRARY

```
import requests, csv, json
from bs4 import BeautifulSoup

def searchbook(bookname):
    titlename = bookname.replace(' ', '+')
    url = "https://link.springer.com/search?new-search=true&query=" + titlename +
    "&content-type=book&dateFrom=&dateTo=&sortBy=relevance"
    return url

def recomend(book, isbnlist, b):
    response2 = requests.get(book)
    soup2 = BeautifulSoup(response2.content, "html.parser")
    titles2 = [title for title in soup2.find_all('a',
        {'class': 'c-article-recommendations-card__link'})]
    a1 = soup2.find('h1').get_text().strip()
    a2 = soup2.find('p', {'data-test': 'bibliographic-information_pisbn'})
    a3 = soup2.find('p', {'class': 'app-article-metrics-bar__count'})
    if(a2):
        b2 = a2.find_all('span')
        a2 = b2[1].get_text().replace('-', '')
    else:
        a2 = book[-17:].replace('-', '')
    isbnlist.append([a1, a2, a3.get_text().strip()[:-8], book])
    for title in titles2:
        b.append([title.get_text().strip(), title.get_attribute_list('href')[0]])
    return 0

def rating(isbn, i, isbnlist):
    url3 = "https://openlibrary.org/search.json?isbn=" + isbn + "&fields=rating*"
    response3 = requests.get(url3)
    jdata = json.loads(response3.content)
    data2 = jdata['docs']
    if(len(data2) > 0 and len(data2[0]) > 0):
        data3 = data2[0]
        isbnlist[i].extend([data3['ratings_average'], data3['ratings_sortable'],
            data3['ratings_count_1'], data3['ratings_count_2'],
            data3['ratings_count_3'], data3['ratings_count_4'],
            data3['ratings_count_5'], data3['ratings_count']])
    else:
        isbnlist[i].extend([0, 0, 0, 0, 0, 0, 0])

def searchrelatebook(bookname):
    url = searchbook(bookname)
    response = requests.get(url)
    soup = BeautifulSoup(response.content, "html.parser")
    titles = [title for title in soup.find_all("h3")]
    a = titles[0].find('a')['href']
    b = [titles[0].get_text(), "https://link.springer.com" + a]
    isbnlist = []
    for i in range(6):
        recomend(b[i][1], isbnlist, b)
    for i in range(6):
        rating(isbnlist[i][1], i, isbnlist)

    file = open('export_data.csv', 'a', newline='', encoding='utf-8')
    writer = csv.writer(file)
    headers = ['title', 'isbn', 'access', 'link', 'ratings_avg',
        'ratings_sortable', 'ratings_count_1', 'ratings_count_2',
        'ratings_count_3', 'ratings_count_4', 'ratings_count_5',
        'ratings_count']
    writer.writerow(headers)
    for title in isbnlist:
        headers = (title)
        writer.writerow(headers)
    file.close()

searchrelatebook("Radiology in Global Health")
```

# AI/ML COMPONENT

```
import pandas as pd
import re
import string
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

df["combined_text"] = df["organizations"] + " classifications: " + df["classifications"] + \
                     " publish_name: " + df["publish_name"] + " auth-keywords: " + df["auth-keywords"] + \
                     " affiliations: " + df['affiliations']

def clean_text(text):
    text = re.sub(r"\[\\"\\]", "", str(text))
    text = re.sub(r"\s+", " ", text)
    return text.strip()

df["combined_text"] = df["combined_text"].apply(clean_text)

def normalize_text(text):
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))
    return text

df["combined_text"] = df["combined_text"].apply(normalize_text)

label_encoder = LabelEncoder()
df["group_subjectEncoded"] = label_encoder.fit_transform(df["grouped_subject"])
num_classes = len(label_encoder.classes_)

X = df["combined_text"]
y = to_categorical(df["group_subjectEncoded"], num_classes=num_classes)
```

# AI/ML COMPONENT

```
X_train = X_train.to_numpy()  
X_test = X_test.to_numpy()  
  
vectorizer = tf.keras.layers.TextVectorization(max_tokens=1000, output_sequence_length=100)  
vectorizer.adapt(X_train)  
  
X_train_vec = vectorizer(X_train)  
X_test_vec = vectorizer(X_test)  
  
X_train_vec = np.array(X_train_vec)  
X_test_vec = np.array(X_test_vec)
```

# AI/ML COMPONENT

```
model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train_vec.shape[1],)),
    BatchNormalization(),
    Dropout(0.4),

    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='relu'),
    Dropout(0.3),

    Dense(num_classes, activation='softmax')
])

model.compile(
    optimizer='adamax',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-5)

history = model.fit(
    X_train_vec, y_train,
    validation_data=(X_test_vec, y_test),
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping, reduce_lr]
)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train_vec, y_train, validation_data=(X_test_vec, y_test), epochs=1, batch_size=32)
```

# VIZ COMPONENT

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import TextVectorization
from sklearn.preprocessing import LabelEncoder
import re
import string
import pickle

def preprocess_input(book_info):
    def clean_text(text):
        text = re.sub(r"\[\]", "", str(text))
        text = re.sub(r"\s+", " ", text)
        return text.strip()

    book_info = clean_text(book_info)

    def normalize_text(text):
        text = text.lower()
        # Remove punctuation
        text = text.translate(str.maketrans("", "", string.punctuation))
        return text

    book_info = normalize_text(book_info)

    # Preprocess the book information (e.g., title, abstract)
    vectorizer = tf.keras.layers.TextVectorization(max_tokens=1000, output_sequence_length=100)
    vectorizer.adapt(np.array([book_info]))
    new_input_vectorized = vectorizer(np.array([book_info])) # Wrap in np.array to match expected input format

    return np.array(new_input_vectorized)
```

# VIZ COMPONENT

```
# Function to predict the subject
def predict_subject(publich_name, organizations, classifications, auth_keywords):
    feature_string = organizations + " classifications: " + classifications + \
    |   |   |   |   " publish_name: " + publich_name + " auth-keywords: " + auth_keywords

    # Preprocess the book information
    processed_input = preprocess_input(feature_string)

    # Predict the probabilities for each subject
    predictions = model.predict(processed_input)

    # Get the index of the highest probability
    best_subject_index = np.argmax(predictions)

    # Map index to class name (your subject labels)
    predicted_subject = label_encoder.inverse_transform([best_subject_index])[0]

    return predicted_subject
```

# VIZ COMPONENT

```
# Load the model
model = load_model('my_dnn_model.keras')

# Load the LabelEncoder
with open('label_encoder.pkl', 'rb') as file:
    label_encoder = pickle.load(file)

-----
# Streamlit things
# Streamlit Layout
st.title("Research Subject Prediction & Book Recommendation Dashboard")

# Input Section
st.sidebar.header("Input for Prediction")
book_publich_name = st.sidebar.text_input("Enter Publish Name")
book_organizations = st.sidebar.text_input("Enter Organizations")
book_classifications = st.sidebar.text_input("Enter Classifications (you can input more than 1 classification)")
auth_keywords = st.sidebar.text_input("Enter Author Keywords (you can input more than 1 keyword)")
```

# VIZ COMPONENT

```
if st.sidebar.button("Predict Subject"):
    if not (book_publich_name or book_organizations or book_classifications or auth_keywords):
        st.sidebar.error("Please fill at least one field before prediction.")
        predicted_subject = None
    else:
        predicted_subject = predict_subject(book_publich_name, book_organizations, book_classifications, auth_keywords)
        if predicted_subject:
            st.sidebar.success(f"Predicted Subject: {predicted_subject}")
    else:
        predicted_subject = None

# Visualization Section
if predicted_subject:
    st.header(f"Books recommended for Subject: {predicted_subject}")
```

# VIZ COMPONENT

```
try:  
    books_df = pd.read_csv(f'{predicted_subject}.csv')  
  
    total_books = len(books_df)  
    st.write(f"Displaying {total_books} books for subject: {predicted_subject}")  
  
    # Table with book details  
    st.dataframe(books_df[['title', 'author_name', 'href', 'ratings_avg', 'ratings_count', 'readinglog_count']])  
  
    # Chart avg rating  
    st.subheader("Book Ratings Distribution")  
    fig, ax = plt.subplots(figsize=(10, 6))  
    ax.hist(books_df['ratings_avg'], bins=20, color='skyblue', edgecolor='black')  
    ax.set_title("Distribution of Average Ratings", fontsize=16, fontweight='bold', color='darkblue')  
    ax.set_xlabel("Average Rating", fontsize=12, fontweight='bold', color='darkgreen')  
    ax.set_ylabel("Number of Books", fontsize=12, fontweight='bold', color='darkgreen')  
    ax.grid(True, axis='y', linestyle='--', alpha=0.7)  
    ax.set_xticks(np.arange(1, 6, 0.5))  
    ax.tick_params(axis='both', labelsize=10)  
    st.pyplot(fig)
```

# VIZ COMPONENT

```
# Chart total views
st.subheader("Book Views Distribution")
fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(books_df['readinglog_count'], bins=20, color='lightgreen', edgecolor='black')
ax.set_title("Distribution of Book Views", fontsize=16, fontweight='bold', color='darkblue')
ax.set_xlabel("Number of Views")
ax.set_ylabel("Number of Books")
ax.grid(True, axis='y', linestyle='--', alpha=0.7)
ax.set_xticks(np.arange(0, 2000, 100))
ax.tick_params(axis='both', labelsize=10)
st.pyplot(fig)

except FileNotFoundError:
    st.error(f"No CSV file found for subject: {predicted_subject}")
```

**DEMO  
TIME**