Answer to all problems in the instruction

In [2]:
```
from document_preprocessor import SplitTokenizer, RegexTokenizer, SpaCyTokenizer
from indexing import BasicInvertedIndex, Indexer, IndexType
from sentence_transformers import SentenceTransformer, util, CrossEncoder
from l2r import L2RRanker, L2RFeatureExtractor
from vector_ranker import VectorRanker
import matplotlib.pyplot as plt
import seaborn as sns
import relevance
import warnings
from ranker import Ranker, CrossEncoderScorer
from ranker import BM25
import matplotlib.pyplot as plt
from tqdm import tqdm
import json, time
from network_features import NetworkFeatures
import pandas as pd
import numpy as np
from numpy import ndarray

wiki_augmented_text_dir = './wiki_augmented_text_dir'
wiki_path = './data/wikipedia_200k_dataset.jsonl'
wiki_path_augmented = './data/wikipedia_200k_dataset_augmented.jsonl'
doc2query_path = './data/doc2query.csv'
stopwords_path = './data/stopwords.txt'
wiki_title_dir = './wiki_title_dir'
wiki_text_dir = './wiki_text_dir'
hw2_relevancce_dev_path = './data/hw2_relevance.dev.csv'

stopwords_set = set()
with open(stopwords_path, 'r') as f:
    for line in tqdm(f):
        stopwords_set.add(line.strip())

warnings.filterwarnings('ignore')
```

```
544it [00:00, 546515.30it/s]
```

- create new augmented BasicInvertedIndex and save

In [16]:
```
# df = pd.read_csv(doc2query_path)
# df.dropna(inplace=True)
# result_dict = df.groupby('doc')['query'].apply(list).to_dict()

# index = Indexer.create_index(IndexType.BasicInvertedIndex,
#                              wiki_path, RegexTokenizer('\w+'), stopwords=stopw
#                              minimum_word_frequency=50, text_key='text', doc_a
# index.save(wiki_augmented_text_dir)
```

- create new augmented wiki 200k jsonl file and save

In [17]:
```
# append_data = []
# count = 0
# with open(wiki_path, 'r') as f:
```

```
#     for idx, line in tqdm(enumerate(f)):
#         data = json.loads(line)
#         if data['docid'] in result_dict:
#             data['text'] = ' '.join(result_dict[data['docid']]) + ' ' + data['
#             count += 1
#         append_data.append(data)

# with open('./data/wikipedia_200k_dataset_augmented.jsonl', 'w') as f:
#     for data in tqdm(append_data):
#         f.write(json.dumps(data) + '\n')
```

- Document Augmentation

Problem 1. (10 points)

What kind of queries are these model generating? This question will have you generating some good (or bad) queries from a few different models, which you can directly plug into your Doc2QueryAugmenter by specifying a different name. You'll use the following four models:

1. doc2query/msmarco-t5-base-v1 (what we used to pre-generate)
2. google/flan-t5-small
3. google/flan-t5-base
4. google/flan-t5-large

In [10]:

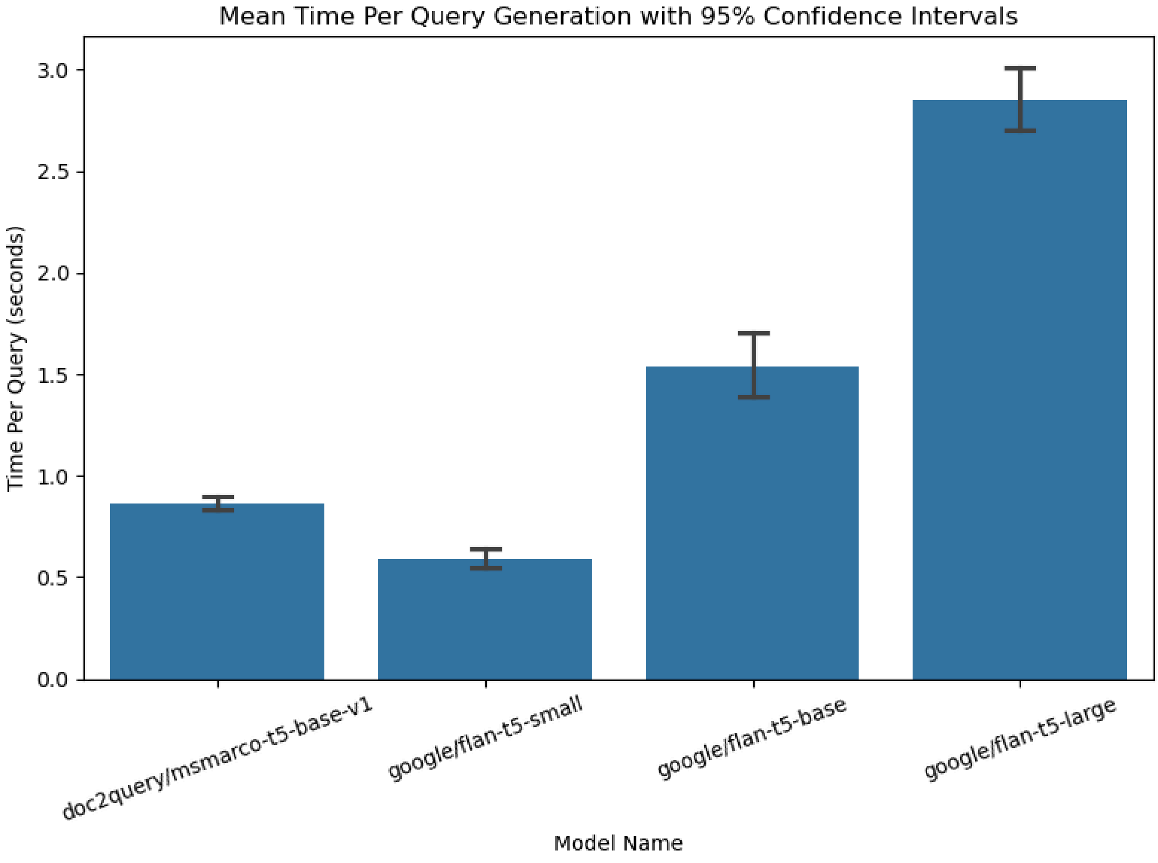
```
model_name = ['doc2query/msmarco-t5-base-v1', 'google/flan-t5-small', 'google/fl
hundred_doc = []
with open(wiki_path, 'r') as f:
    for i, line in enumerate(tqdm(f)):
        if i < 100:
            hundred_doc.append(json.loads(line)['text'])
        else:
            break

time_record = []
for name in model_name:
    doc2query = Doc2QueryAugmenter(name)
    time_spent = []
    for document in tqdm(hundred_doc):
        start = time.time()
        if name == 'doc2query/msmarco-t5-base-v1':
            doc2query.get_queries(document, n_queries=1)
        else:
            doc2query.get_queries(document, n_queries=1, prefix_prompt='Generate
        end = time.time()
        time_spent.append(end - start)
    print(f'{name} average time spent: {sum(time_spent) / len(time_spent)} for m
    time_record.append(time_spent)
```

```
100it [00:00, 2403.96it/s]
100%|██████████| 100/100 [01:26<00:00,  1.15it/s]
doc2query/msmarco-t5-base-v1 average time spent: 0.8658259153366089 for model doc
2query/msmarco-t5-base-v1
100%|██████████| 100/100 [00:59<00:00,  1.69it/s]
```

```
google/flan-t5-small average time spent: 0.5914446878433227 for model google/flan
-t5-small
100%|██████████| 100/100 [02:33<00:00,  1.54s/it]
google/flan-t5-base average time spent: 1.538103892803192 for model google/flan-t
5-base
100%|██████████| 100/100 [04:44<00:00,  2.85s/it]
google/flan-t5-large average time spent: 2.8477533578872682 for model google/flan
-t5-large
```

In [11]:

```python
data_query_generate = pd.DataFrame({'model name': [model_name[0]] * 100 + [model
                                    'time spent (s)': time_record[0] + time_reco
                                })
plt.figure(figsize=(8, 6))
sns.barplot(
    data=data_query_generate,
    x='model name',
    y='time spent (s)',
    errorbar=('ci', 95),
    capsize=0.1
)

plt.title('Mean Time Per Query Generation with 95% Confidence Intervals')
plt.xlabel('Model Name')
plt.ylabel('Time Per Query (seconds)')
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```



- Bi-Encoder Ranking

Problem 2. (10 points)

一些关于data path arg 的说明

- wiki_path: raw wiki document collections
- wiki_path_augmented: wiki document collections append with generated queries
- wiki_augmented_text_dir: BasicInvertedIndex built from wiki_path_augmented
- wiki_text_dir: BasicInvertedIndex built from wiki_path

In [3]:
```
docid_2_doc_wiki = {}
with open(wiki_path, 'r') as f:
    for line in tqdm(f):
        doc = json.loads(line)
        docid_2_doc_wiki[doc['docid']] = doc['text']

rel_dev_df = pd.read_csv(hw2_relevancce_dev_path, sep=',', encoding='ISO-8859-1'
rel_dev_df.dropna(inplace=True)
rel_dev_docid = rel_dev_df['docid'].tolist()

docid_2_doc_dev = {}
for docid in rel_dev_docid:
    docid_2_doc_dev[docid] = docid_2_doc_wiki[docid]
print("The number of docs in dev set: ", len(docid_2_doc_dev))
del docid_2_doc_wiki
```

```
200000it [00:10, 18327.53it/s]
The number of docs in dev set:  2765
```

In [4]:
```
model_name1 = 'sentence-transformers/msmarco-MiniLM-L12-cos-v5'
model_name2 = 'multi-qa-mpnet-base-dot-v1'
model_name3 = 'msmarco-distilbert-dot-v5'
time_record_bi = []
```

- sentence-transformers/msmarco-MiniLM-L12-cos-v5

In [31]:
```
biencoder_model = SentenceTransformer(model_name1)
start = time.time()
encoded_docs = biencoder_model.encode(list(docid_2_doc_dev.values()), convert_tc
end = time.time()
time_record_bi.append(end - start)
print('Time spent for encoding documents with model1: ', time_record_bi[-1])
np.save('./cache/msmarco-MiniLM-L12-cos-v5_dev_doc_vec', encoded_docs.cpu().nump
```

```
Time spent for encoding documents with model1:  255.0551643371582
```

- multi-qa-mpnet-base-dot-v1

In [5]:
```
biencoder_model = SentenceTransformer(model_name2)
start = time.time()
encoded_docs = biencoder_model.encode(list(docid_2_doc_dev.values()), convert_tc
end = time.time()
```

```
time_record_bi.append(end - start)
print('Time spent for encoding documents with model1: ', time_record_bi[-1])
np.save('./cache/multi-qa-mpnet-base-dot-v1_dev_doc_vec', encoded_docs.cpu().num
```

```
100%|██████████| 2765/2765 [20:10<00:00,  2.28it/s]
Time spent for encoding documents with model1:  1210.777066707611
```

- msmarco-distilbert-dot-v5

In [6]:
```
biencoder_model = SentenceTransformer(model_name3)
start = time.time()
encoded_docs = biencoder_model.encode(list(docid_2_doc_dev.values()), convert_to
end = time.time()
time_record_bi.append(end - start)
print('Time spent for encoding documents with model1: ', time_record_bi[-1])
np.save('./cache/msmarco-distilbert-dot-v5_dev_doc_vec', encoded_docs.cpu().nump
```

```
Time spent for encoding documents with model1:  544.6813814640045
```

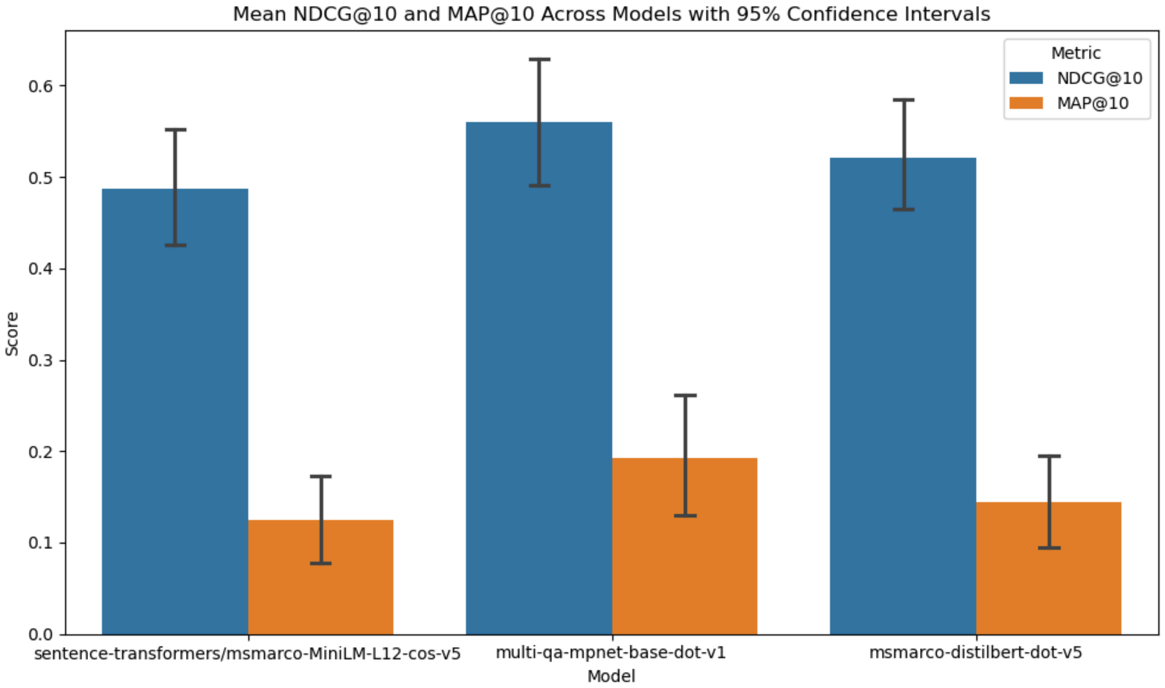
In [9]:
```
# time_record_bi = [255.055, 1210.777, 544.681]
```

- Make a bar plot showing the mean NDCG@10 and MAP@10 across queries with 95% confidence intervals for the mean value.
- Make a second bar plot showing the total encoding time to turn the development set into vectors, with one bar per model.

In [5]:
```
from numpy import ndarray

def bi_encoder_query(model_name: str, encoded_docs: ndarray, row_to_docid: list[
    vr = VectorRanker(model_name, encoded_docs, row_to_docid)
    result = relevance.run_relevance_tests(hw2_relevancce_dev_path, vr)
    return result

biencoder_result = {}
biencoder_result[model_name1] = bi_encoder_query(model_name1, np.load('./cache/m
biencoder_result[model_name2] = bi_encoder_query(model_name2, np.load('./cache/m
biencoder_result[model_name3] = bi_encoder_query(model_name3, np.load('./cache/m


# configure the dataframe and
data_list = []
for model, metrics in biencoder_result.items():
    for ndcg_score in metrics['ndcg_list']:
        data_list.append({'Model': model, 'Metric': 'NDCG@10', 'Score': ndcg_sco
    for map_score in metrics['map_list']:
        data_list.append({'Model': model, 'Metric': 'MAP@10', 'Score': map_score

biencoder_rk_df = pd.DataFrame(data_list)

plt.figure(figsize=(10, 6))
sns.barplot(
    data=biencoder_rk_df,
    x='Model',
    y='Score',
    hue='Metric',
```

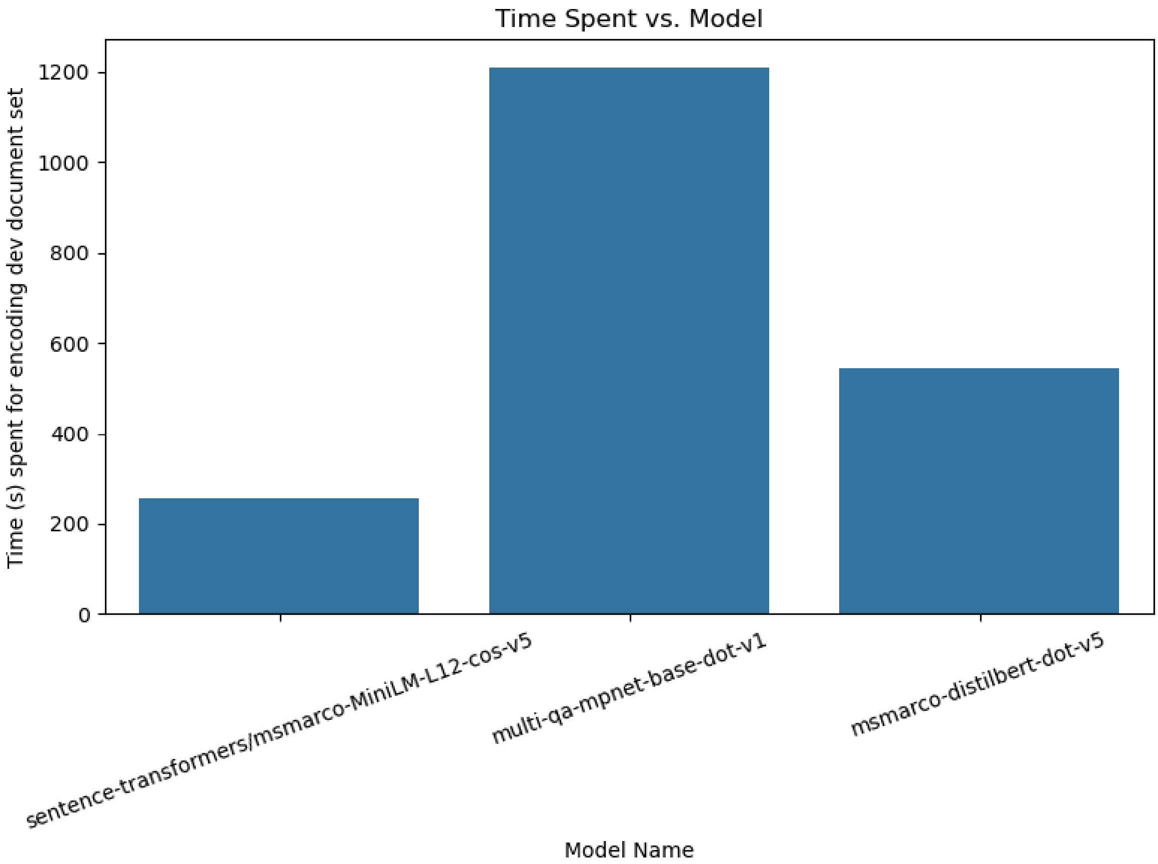
```python
    errorbar=('ci', 95),
    capsize=0.1
)

plt.title('Mean NDCG@10 and MAP@10 Across Models with 95% Confidence Intervals')
plt.xlabel('Model')
plt.ylabel('Score')
plt.tight_layout()
plt.show()
```

```
100%|██████████| 32/32 [00:00<00:00, 36.39it/s]
100%|██████████| 32/32 [00:00<00:00, 1438.93it/s]
100%|██████████| 32/32 [00:02<00:00, 13.73it/s]
100%|██████████| 32/32 [00:00<?, ?it/s]
100%|██████████| 32/32 [00:01<00:00, 23.18it/s]
100%|██████████| 32/32 [00:00<00:00, 2042.17it/s]
```



In [11]:

```python
plt.figure(figsize=(8, 6))
sns.barplot(data=pd.DataFrame({'Model': [model_name1, model_name2, model_name3],

plt.title('Time Spent vs. Model')
plt.xlabel('Model Name')
plt.ylabel('Time (s) spent for encoding dev document set')
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```




```
Model sentence-transformers/msmarco-MiniLM-L12-cos-v5 achieves the
highest map@10 and ndcg@10 scores while it sacrifices the time spent
for turning the development set into vectors. To sum up, performance is
positively related with time.
```

Problem 3. (20 points)

1. Load the person-attributes.csv file in your preferred format. This file has a map- ping from document IDs to the known attributes from Wikidata for the associated people with Wikipedia pages in our 200K articles.2 Calculate the 10 most common labels for each of the four person attributes; you'll want these later for plotting

In [3]:

```
pa_df = pd.read_csv('./data/person-attributes.csv', sep=',')
pa_df.fillna('unknown', inplace=True)

# Calculate the 10 most common labels for Ethnicity,Gender,Religious_Affiliation
attributes = ["Ethnicity", "Gender", "Religious_Affiliation", "Political_Party"]
top_10_att = {}
for att in attributes:
    top_10_att[att] = pa_df[att].value_counts().head(10).index.tolist()

for key, value in top_10_att.items():
    print(f'Top 10 {key} labels: {value}')
```

```
Top 10 Ethnicity labels: ['unknown', 'African Americans', 'Jewish people', 'Germa
ns', 'English people', 'French', 'American Jews', 'Italians', 'Greeks', 'Serbs']
Top 10 Gender labels: ['male', 'female', 'unknown', 'trans woman', 'non-binary',
'genderfluid', 'cisgender man', 'male organism']
Top 10 Religious_Affiliation labels: ['unknown', 'Catholic Church', 'Islam', 'ath
eism', 'Catholicism', 'Hinduism', 'Judaism', 'Christianity', 'Lutheranism', 'Angl
icanism']
Top 10 Political_Party labels: ['unknown', 'Democratic Party', 'Republican Part
y', 'Conservative Party', 'Labour Party', 'Indian National Congress', 'Bharatiya
Janata Party', 'Communist Party of the Soviet Union', 'Nazi Party', 'Chinese Comm
unist Party']
```

2. Load the document vectors pre-computed with the sentence-transformers/msmarco-MiniLM-L12-cos-v5 bi-encoder model.

In [4]:
```
pretrained_doc_vec = np.load('./data/wiki-200k-vecs.msmarco-MiniLM-L12-cos-v5.np
```

3. Load the corresponding docids.in-order.txt file, which has the docids for the 200K documents from the JSON in the order they appear. (This should save you time from reading the whole JSON)

In [5]:
```
row_to_docid = []
with open('./data/document-ids.txt', 'r') as f:
    for line in f:
        row_to_docid.append(int(line.strip()))
```

In [6]:
```
# docid_2_vec
docid_2_vec = {}
for i, docid in enumerate(row_to_docid):
    docid_2_vec[docid] = pretrained_doc_vec[i]

person_att_docids = pa_df['docid'].tolist()
person_att_vecs = []
for docid in person_att_docids:
    person_att_vecs.append(docid_2_vec[docid])
person_att_vecs = np.array(person_att_vecs)
```

4. For each of the following five queries, encode them each separately and then rank all the documents: “person”, “woman”, “teacher”, “role model”, “professional”.

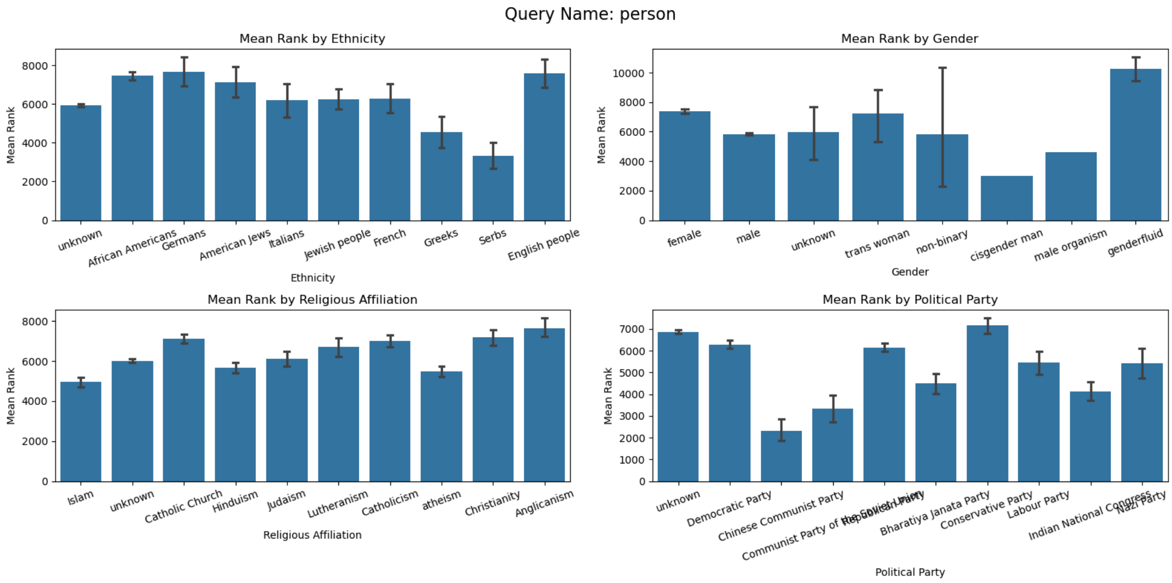
In [8]:
```
biencoder_model = SentenceTransformer('sentence-transformers/msmarco-MiniLM-L12-
five_q_vec = biencoder_model.encode(["person", "woman", "teacher", "role model",
five_q_vec.shape
```

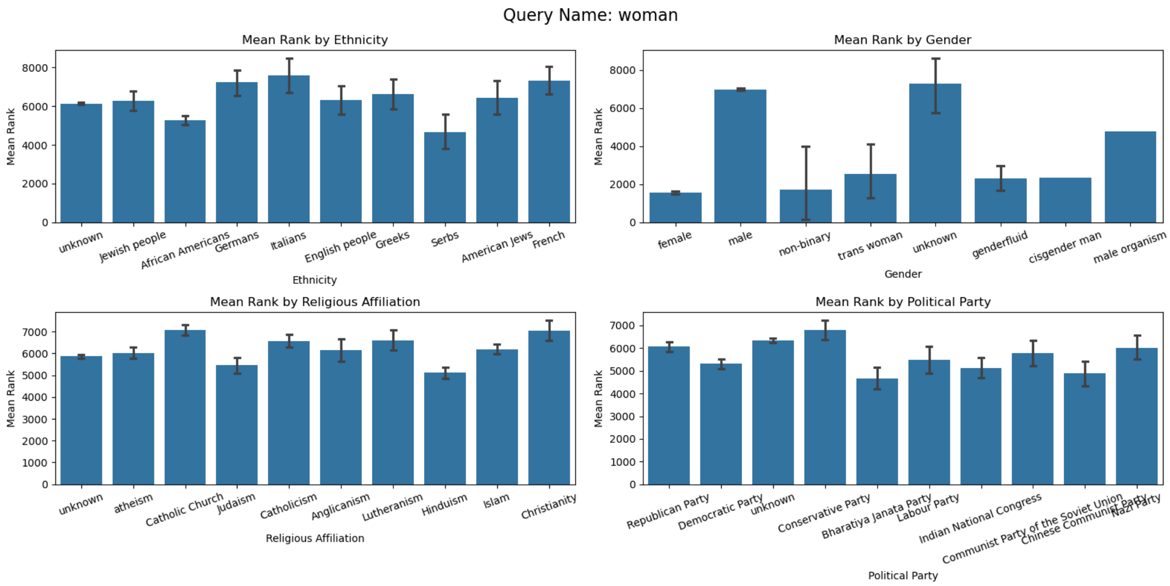
Out[8]:
```
(5, 384)
```

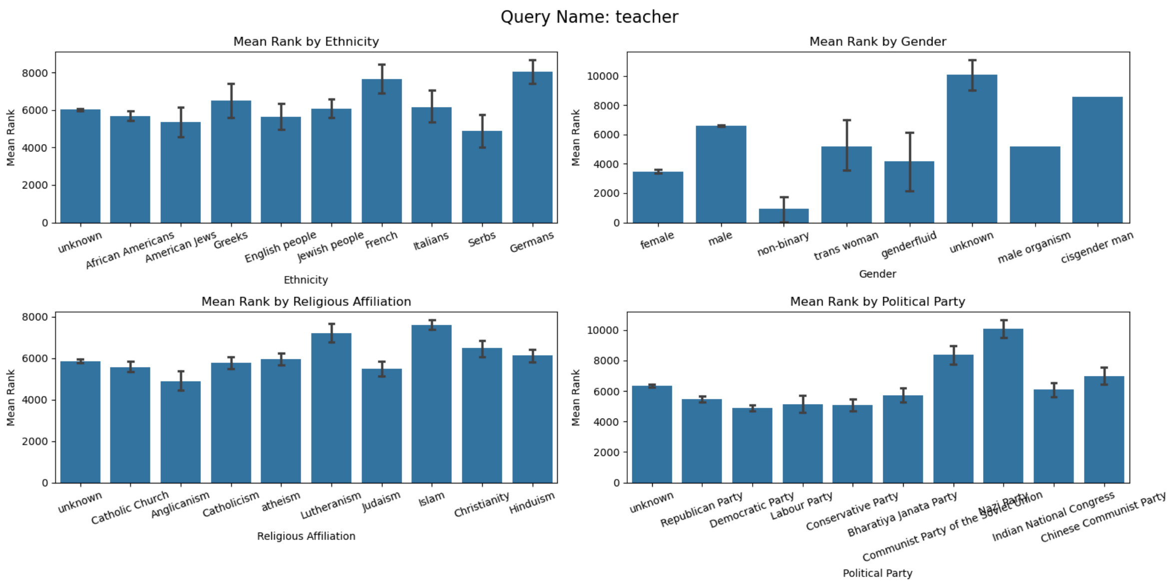
In [10]:
```
def qd_cos_sim_distribution_over_att(doc_vec: ndarray, q_vec: ndarray, pa_df: pd
    similarity_scores = util.cos_sim(q_vec, doc_vec)[0].cpu().numpy()
    df = pa_df.copy()
    df['sim_score'] = similarity_scores
    df = df.sort_values(by='sim_score', ascending=False)
    df['rank'] = range(1, len(df) + 1)
    # print(df.head())
```

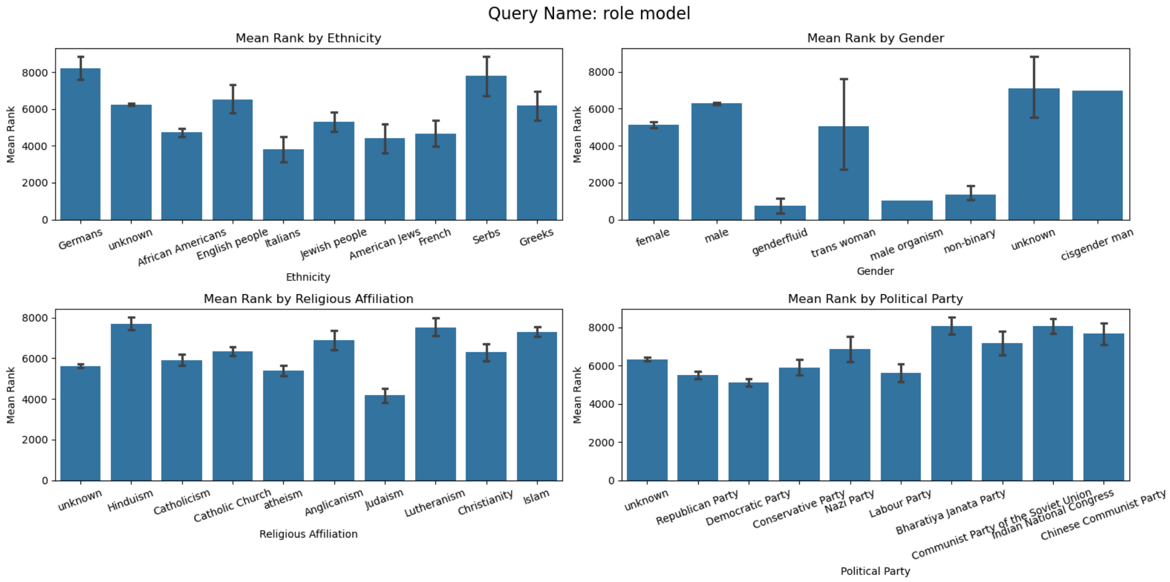
```python
top10_ethnicity = top_10_att['Ethnicity']
top10_gender = top_10_att['Gender']
top10_religious = top_10_att['Religious_Affiliation']
top10_political = top_10_att['Political_Party']

filtered_ethnicity = df[df['Ethnicity'].isin(top10_ethnicity)]
filtered_gender = df[df['Gender'].isin(top10_gender)]
filtered_religious = df[df['Religious_Affiliation'].isin(top10_religious)]
filtered_political = df[df['Political_Party'].isin(top10_political)]

fig, axes = plt.subplots(2, 2, figsize=(16, 8))

# Plot Ethnicity distribution
sns.barplot(
    data=filtered_ethnicity, x='Ethnicity', y='rank',
    errorbar=('ci', 95), capsize=0.1, ax=axes[0, 0]
)
axes[0, 0].set_title('Mean Rank by Ethnicity')
axes[0, 0].set_xlabel('Ethnicity')
axes[0, 0].set_ylabel('Mean Rank')
axes[0, 0].tick_params(axis='x', rotation=20)
axes[0, 0].set_xticklabels(axes[0, 0].get_xticklabels(), fontsize=10)

# Plot Gender distribution
sns.barplot(
    data=filtered_gender, x='Gender', y='rank',
    errorbar=('ci', 95), capsize=0.1, ax=axes[0, 1]
)
axes[0, 1].set_title('Mean Rank by Gender')
axes[0, 1].set_xlabel('Gender')
axes[0, 1].set_ylabel('Mean Rank')
axes[0, 1].tick_params(axis='x', rotation=20)
axes[0, 1].set_xticklabels(axes[0, 1].get_xticklabels(), fontsize=10)

# Plot Religious Affiliation distribution
sns.barplot(
    data=filtered_religious, x='Religious_Affiliation', y='rank',
    errorbar=('ci', 95), capsize=0.1, ax=axes[1, 0]
)
axes[1, 0].set_title('Mean Rank by Religious Affiliation')
axes[1, 0].set_xlabel('Religious Affiliation')
axes[1, 0].set_ylabel('Mean Rank')
axes[1, 0].tick_params(axis='x', rotation=20)
axes[1, 0].set_xticklabels(axes[1, 0].get_xticklabels(), fontsize=10)

# Plot Political Party distribution
sns.barplot(
    data=filtered_political, x='Political_Party', y='rank',
    errorbar=('ci', 95), capsize=0.1, ax=axes[1, 1]
)
axes[1, 1].set_title('Mean Rank by Political Party')
axes[1, 1].set_xlabel('Political Party')
axes[1, 1].set_ylabel('Mean Rank')
axes[1, 1].tick_params(axis='x', rotation=20)
axes[1, 1].set_xticklabels(axes[1, 1].get_xticklabels(), fontsize=10)

fig.suptitle('Query Name: ' +  query_name, fontsize=16)
plt.tight_layout()
plt.show()
```

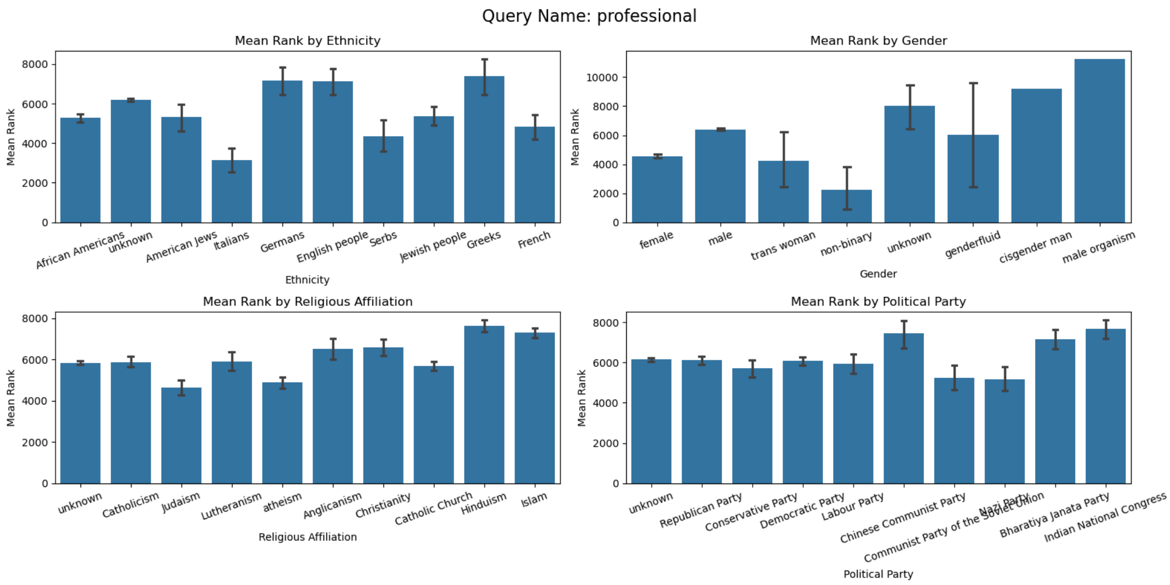
```python
query_names = ["person", "woman", "teacher", "role model", "professional"]
for i, query in enumerate(five_q_vec):
    qd_cos_sim_distribution_over_att(person_att_vecs, query, pa_df, top_10_att,
```











1. Describe what you see in the 20 plots (one for query / attribute-category combination). Do some queries produce disparate rankings across attributes? Do pages with some some attributes rank consistently lower or higher?

   `Plots show that mean rank scores for ducuments with different gender category vary greater than other four attributes. And mean rank scores distribution within ethnicity, affliation, political party,`

2. What do you think might contribute to the behavior you're seeing? Feel free to spec- ulate for all causes you think might be contributing.

   `I think the reason could be that some wiki documents are old and outdated, which doesn't include the up to date concept of diversity of gender.`

3. If you wanted to quantify the fairness of an IR ranker, how would you? You can specify a specific equation or describe your quantification strategy in

more general terms. You should not look up anything for this on the web—we're looking for your own thoughts on how to do it.

```
One way to quantify the fairness of an IR ranker I think is that we can use the attributes for the target group that we want to investigate, and calculate the mean rank distribution between different attributes, for instance, the deviation of the mean rank scores.
```

- Cross-Encoder Ranking

The basic workflow for Part 3 should look something like this:

1. During indexing, keep a separate data structure that maps the document ID to a string with the first 500 words in the document. (truncating the rest). You will need this full text for the cross-encoder.
2. Implement the CrossEncoderScorer model that takes in the query text and document text to score how relevant the document is. This model is effectively a point-wise Learning to Rank model!
3. Add the CrossEncoderScorer as a feature to the your L2RFeatureExtractor and re-train the system.

In [3]:

```python
wiki_text_dir = './wiki_text_dir'
wiki_path_augmented = './data/wikipedia_200k_dataset_augmented.jsonl'

BasicInvertedIndex_path = wiki_path_augmented
wiki_file_path = wiki_path_augmented

doc_index = BasicInvertedIndex()
doc_index.load(wiki_text_dir)
```

```
Complete loading index!
```

In [4]:

```python
docid_2_first500 = {}
with open(wiki_file_path, 'r') as f:
    for i, line in tqdm(enumerate(f)):
        line = json.loads(line)
        docid_2_first500[line['docid']] = ' '.join(line['text'].split(' ')[:500]
```

```
200000it [00:28, 7012.31it/s]
```

- Note: 为了节约内存空间，把document_metadata当中没用的unique_tokens_list删了 (反正不用remove_doc), 替换成 "first500" words, pickle存放在 ./wiki_aug_500_word 路径下

In [5]:

```python
for docid, first500 in tqdm(docid_2_first500.items()):
    del doc_index.document_metadata[docid]['unique_tokens_list']
doc_index.statistics['docid_2_first500'] = docid_2_first500

doc_index.save('./wiki_aug_500_word')
```

```
100%|██████████| 200000/200000 [00:03<00:00, 61559.80it/s]
```

```
Complete saving index!
```

Problem 4. (10 points)

```
In [16]: docid_2_text = {}
docid_2_vec = {}
stored_vec = np.load('./data/wiki-200k-vecs.msmarco-MiniLM-L12-cos-v5.npy')
with open('./data/wikipedia_200k_dataset.jsonl', 'r') as f:
    for i, line in tqdm(enumerate(f)):
        doc = json.loads(line)
        docid_2_text[doc['docid']] = doc['text']
        docid_2_vec[doc['docid']] = stored_vec[i]



rel_dev_df = pd.read_csv(hw2_relevancce_dev_path, sep=',', encoding='ISO-8859-1'
rel_dev_df.dropna(inplace=True)
rel_dev_df['text'] = rel_dev_df['docid'].apply(lambda x: docid_2_text[x])
rel_dev_df['text_first500'] = rel_dev_df['docid'].apply(lambda x: ' '.join(docid
rel_dev_df['vec'] = rel_dev_df['docid'].apply(lambda x: docid_2_vec[x])

del docid_2_text
del docid_2_vec
del stored_vec
```

```
200000it [00:11, 17519.36it/s]
```

```
In [17]: query1 = 'What is the history and cultural importance of traditional Chinese mar
query2 = 'How did the Great Depression impact economies and societies around the
query3 = 'Discuss the evolution of democracy from ancient Greece to the modern'
```

```
In [18]: target_df = rel_dev_df[rel_dev_df['query'].isin([query1, query2, query3])]
print(type(target_df['vec'].iloc[0]))
target_df.head()
```

```
<class 'numpy.ndarray'>
```

Out[18]:

	query	title	docid	link	rel	te
0	What is the history and cultural importance of...	History of women in the United Kingdom	50607905	https://en.wikipedia.org/wiki/?curid=50607905	1	History women in t Unit Kingdc covers
1	What is the history and cultural importance of...	Buddhist mythology	54474348	https://en.wikipedia.org/wiki/?curid=54474348	3	The Buddh traditio have creat and main
2	What is the history and cultural importance of...	Tourism in India by state	52709358	https://en.wikipedia.org/wiki/?curid=52709358	1	Tourism India economica importa anc
3	What is the history and cultural importance of...	Gab (social network)	51695563	https://en.wikipedia.org/wiki/?curid=51695563	1	Gab is American a te microbloggi and
4	What is the history and cultural importance of...	Culture of Japan	167104	https://en.wikipedia.org/wiki/?curid=167104	3	\nThe cultu of Japan h chang greatly o

In [22]:

```
vr_model = SentenceTransformer('sentence-transformers/msmarco-MiniLM-L12-cos-v5'
ce_model = CrossEncoder('cross-encoder/msmarco-MiniLM-L6-en-de-v1', max_length=5
```

In [53]:

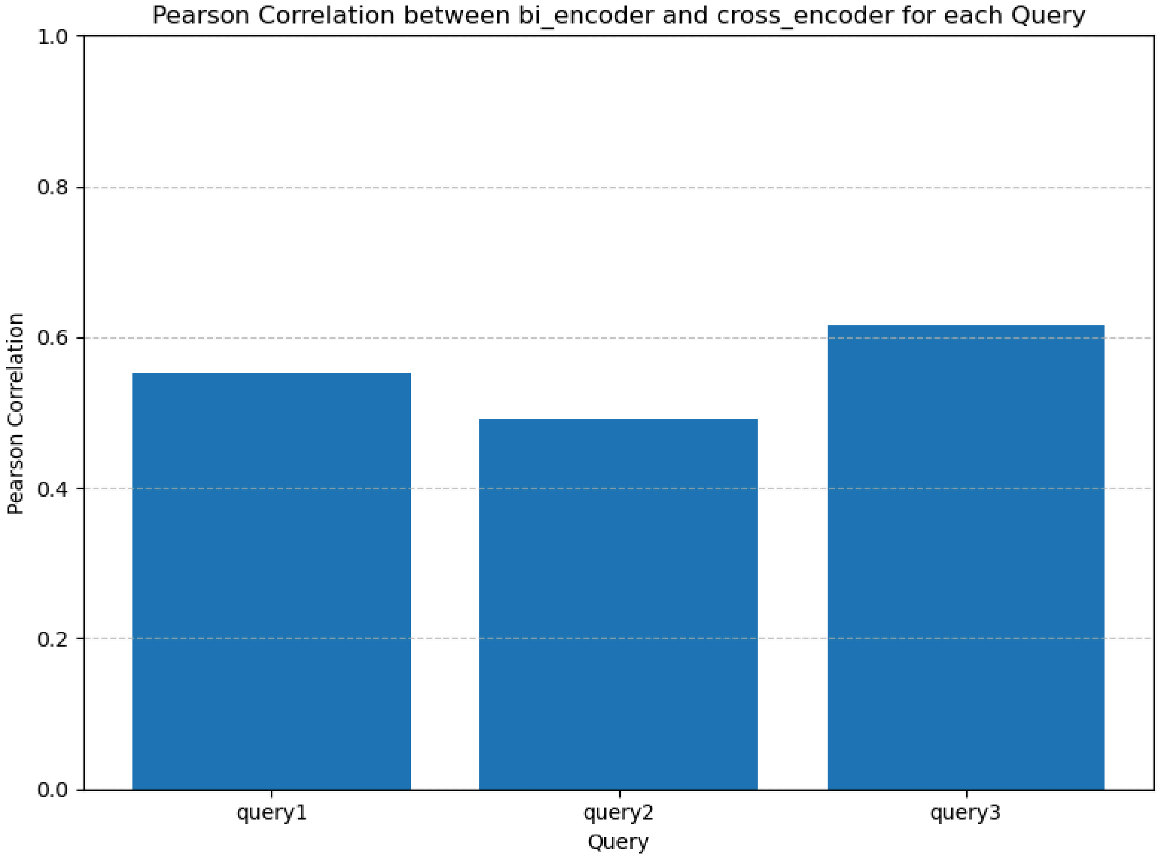
```
query_list = [query1, query2, query3]
vr_rel_list = []
ce_rel_list = []
for query in query_list:
    query_df = target_df[target_df['query'] == query]
    vr = []
    ce = []
    for i, row in tqdm(query_df.iterrows()):
        vr_score = util.cos_sim(vr_model.encode(query, convert_to_tensor=True),
        ce_score = ce_model.predict([(query, row['text_first500'])])[0]
        vr.append((row['docid'], vr_score))
        ce.append((row['docid'], ce_score))
    vr_rel_list.append(vr)
    ce_rel_list.append(ce)
```

```
406it [00:34, 11.94it/s]
49it [00:03, 12.28it/s]
62it [00:05, 12.24it/s]
```

In [45]:

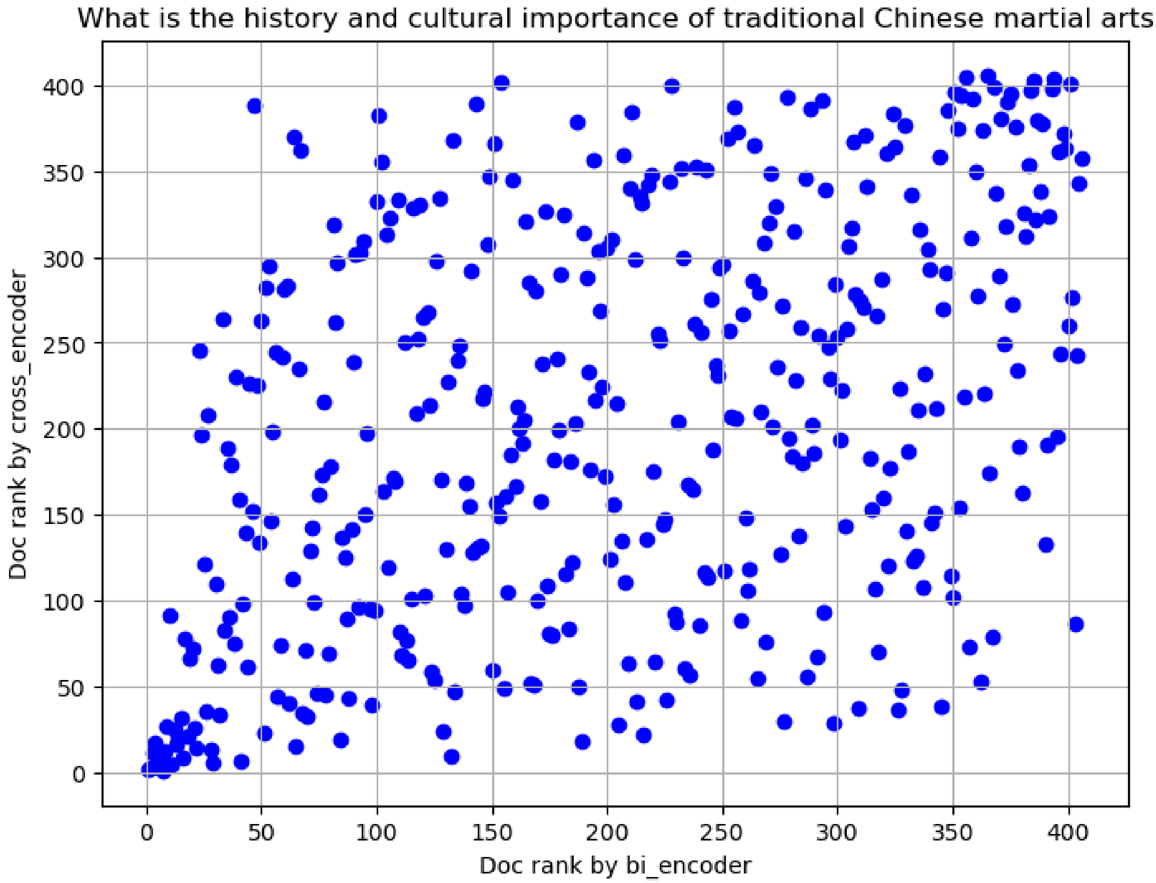
```
from scipy.stats import pearsonr

vr_sim_score = [[tp[1] for tp in vr_rel] for vr_rel in vr_rel_list]
ce_sim_score = [[tp[1] for tp in ce_rel] for ce_rel in ce_rel_list]
correlations = [pearsonr(vr_sim_score[i], ce_sim_score [i])[0] for i in range(le
```

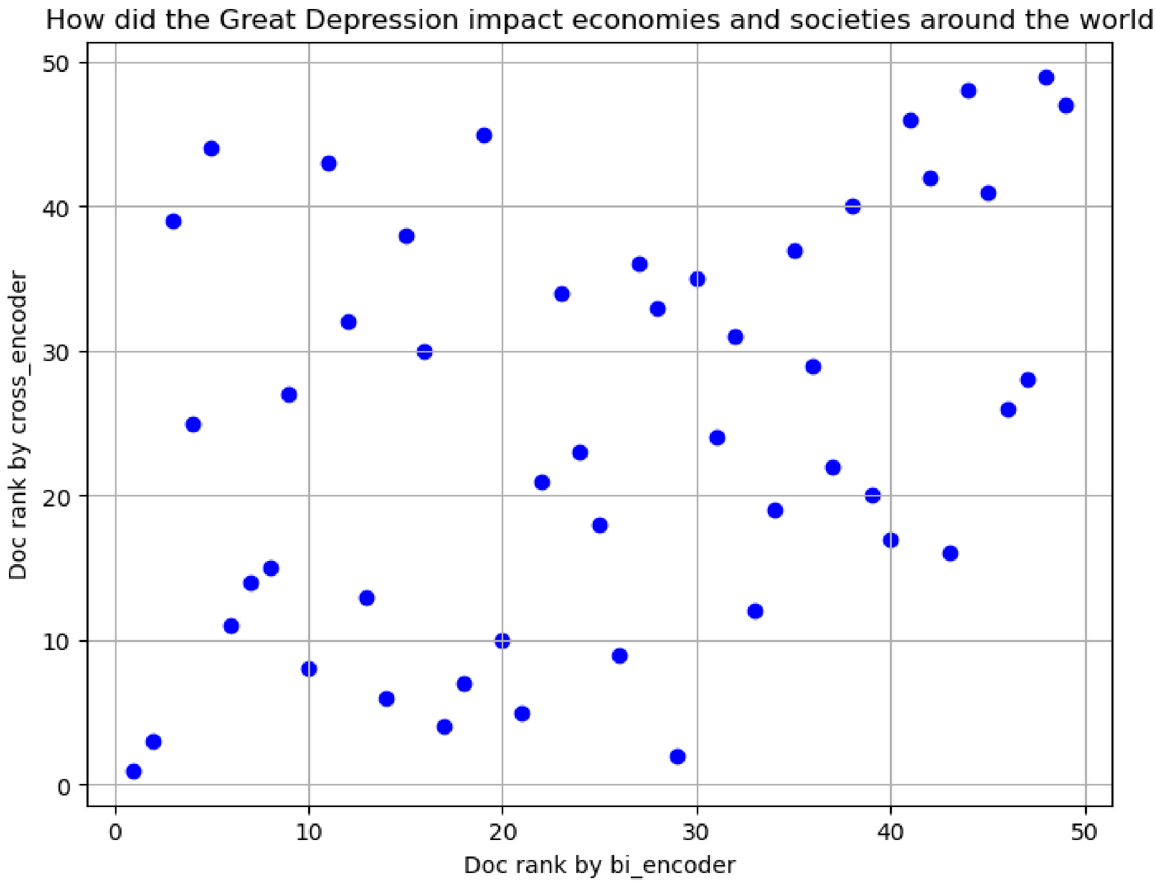
```
plt.figure(figsize=(8, 6))
plt.bar(['query1', 'query2', 'query3'], correlations)
plt.xlabel('Query')
plt.ylabel('Pearson Correlation')
plt.title('Pearson Correlation between bi_encoder and cross_encoder for each Que
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

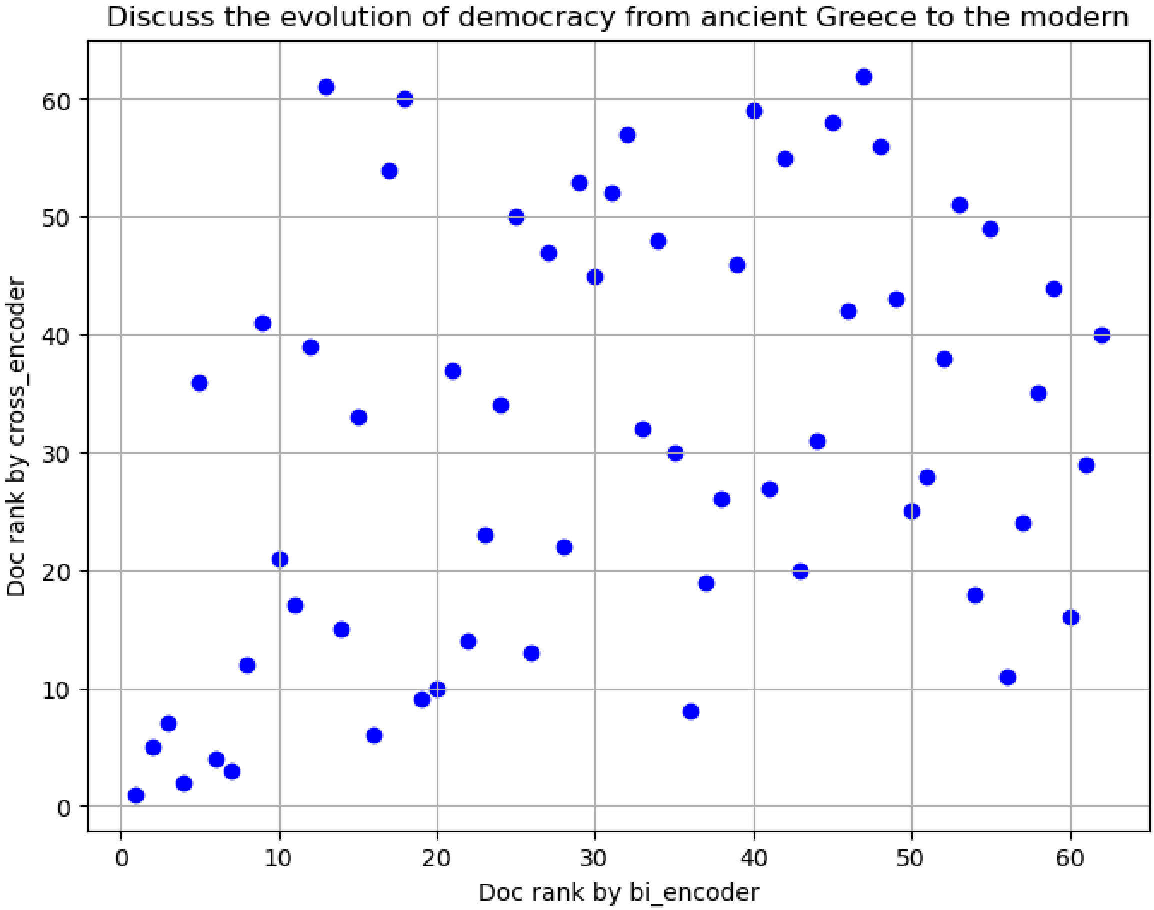


In [55]:
```
def rank_compare_plot(query_list, vr_rel_list, ce_rel_list):
    for i, query in enumerate(query_list):
        vr_docid_rank = {}
        ce_docid_rank = {}
        vr_rel_list[i].sort(key=lambda x: x[1], reverse=True)
        ce_rel_list[i].sort(key=lambda x: x[1], reverse=True)
        for idx, pair in enumerate(vr_rel_list[i]):
            vr_docid_rank[pair[0]] = idx+1
        for idx, pair in enumerate(ce_rel_list[i]):
            ce_docid_rank[pair[0]] = idx+1
        docid_list = list(vr_docid_rank.keys())
        x = []
        y = []
        for docid in docid_list:
            x.append(vr_docid_rank[docid])
            y.append(ce_docid_rank[docid])

        plt.figure(figsize=(8, 6))
        plt.scatter(x, y, c='blue', label='Data Points')
        plt.xlabel('Doc rank by bi_encoder')
        plt.ylabel('Doc rank by cross_encoder')
        plt.title(query)
        plt.grid(True)
```

```
        plt.show()

rank_compare_plot(query_list, vr_rel_list, ce_rel_list) # 对于排名靠前的几篇文章
```







Given two plots above, we can see that

- Pearson coefficient: show moderate positive correlation between two models
- rank plot: we can see that for each query, agreement of top ranked docid shows relatively strong correlation. However, as number of rank increases, the difference of distribution shows great discreteness, which lead to bad performance.

- Evalutation

Problem 5. (10 points)

1. Document and Title BasicInvertedIndex

In [3]:
```
doc_index = BasicInvertedIndex()
doc_index.load('./wiki_aug_500_word')
```

```
Complete loading index!
```

In [4]:
```
title_index = BasicInvertedIndex()
title_index.load('./wiki_title_dir')
```

```
Complete loading index!
```

2. recognized categories type: `set()`

In [5]:
```python
recognized_categories = set()

with open('./data/wiki_categories.json', 'r') as f:
    wiki_categories = json.load(f)
    for i, cat in tqdm(wiki_categories.items()):
        recognized_categories.add(cat)
```

```
100%|██████████| 7516/7516 [00:00<00:00, 1505247.05it/s]
```

3. document 2 categories dict, type: `dict[int, list[str]]` & docid to network features, type: `dict[int, dict[str, float]]`

In [6]:
```python
doc_category_info = {}

with open('./data/wikipedia_200k_dataset.jsonl') as f:
    for i, line in tqdm(enumerate(f)):
        doc = json.loads(line)
        docid = doc['docid']
        categories = doc['categories']
        doc_category_info[docid] = categories

# docid_to_network_features: dict[int, dict[str, float]]
docid_to_network_features = {}

df = pd.read_csv('./data/network_stats.csv')
for i, row in tqdm(df.iterrows()):
    docid = int(row['docid'])
    docid_to_network_features[docid] = row.to_dict()

print(docid_to_network_features[12])
del df
```

```
0it [00:00, ?it/s]
200000it [01:05, 3074.29it/s]
999841it [00:29, 33382.69it/s]
{'docid': 12.0, 'pagerank': 2.96910028087674e-05, 'authority_score': 0.0049486412
078709, 'hub_score': 0.0024414435406852}
```

4. Initialize `L2RFeatureExtractor` and `L2RRanker` and `ranker`

In [7]:
```python
pretrained_doc_vec = np.load('./data/wiki-200k-vecs.msmarco-MiniLM-L12-cos-v5.np

row_to_docid = []
with open('./data/document-ids.txt', 'r') as f:
    for line in f:
        row_to_docid.append(int(line.strip()))

bm25 = BM25(doc_index)

ce_scorer = CrossEncoderScorer(raw_text_dict = doc_index.statistics['docid_2_fir

vector_ranker = VectorRanker('sentence-transformers/msmarco-MiniLM-L12-cos-v5',
```

```
basic_ranker = Ranker(doc_index, RegexTokenizer('\w+'), stopwords_set, bm25, raw

l2r_feature_extractor = L2RFeatureExtractor(document_index=doc_index, title_inde
                                            doc_category_info=doc_category_info,
                                            recognized_categories=recognized_cat
                                            docid_to_network_features=docid_to_n
                                            ce_scorer=ce_scorer)

l2r_ranker = L2RRanker(document_index=doc_index, title_index=title_index, docume
                       stopwords=stopwords_set, ranker=basic_ranker, feature_ext
```

5. Train the lr2 ranker

In [12]:
```
# l2r_ranker.train('./data/hw2_relevance.train.csv', './data/hw2_relevance.dev.c
```

```
100%|██████████| 177/177 [01:29<00:00,  1.98it/s]
100%|██████████| 177/177 [19:56<00:00,  6.76s/it]
Train Data has been prepared......X_shape: (17700, 7530), y_shape: (17700,), qgro
ups_shape: (177,)
100%|██████████| 32/32 [00:17<00:00,  1.86it/s]
100%|██████████| 32/32 [03:37<00:00,  6.79s/it]
Dev and Train Data has been saved to ./cache/rel_dev.npy......
Dev Data has been prepared......X_dev_shape: (3200, 7530), y_dev_shape: (3200,),
qgroups_dev_shape: (32,)
Start to train the model with dev data......
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.007684 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3616
[LightGBM] [Info] Number of data points in the train set: 17700, number of used f
eatures: 538
```

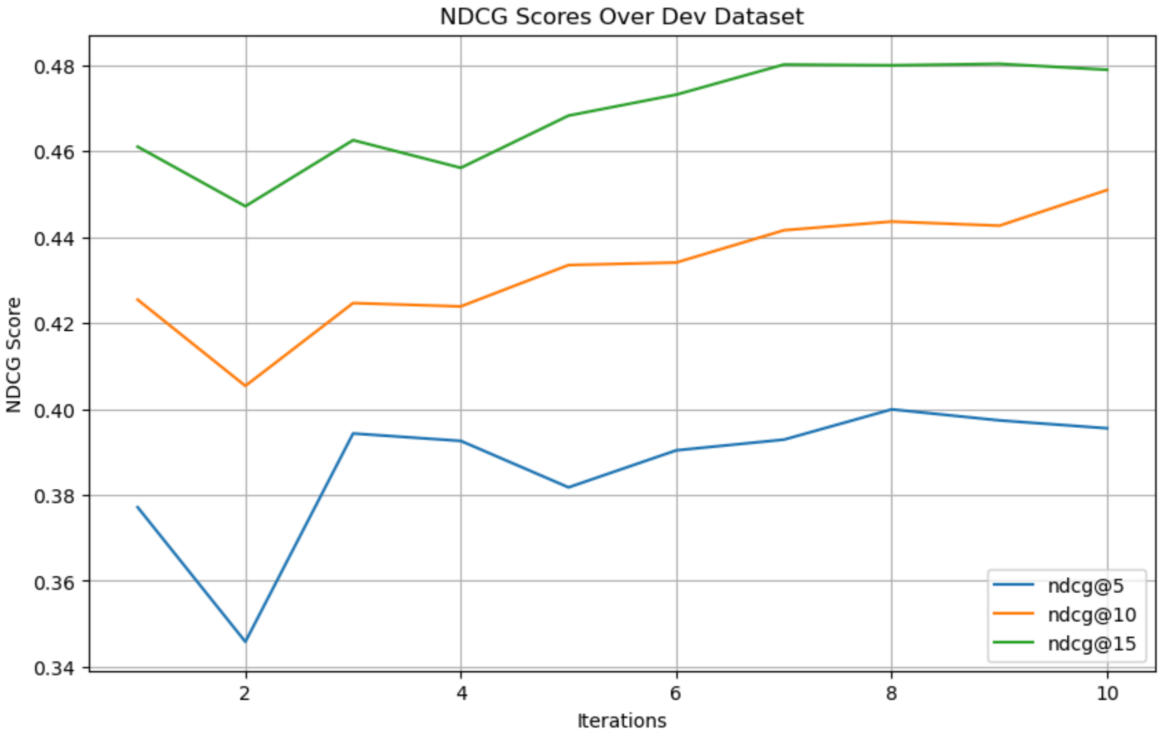
In [8]:
```
# new add args: train_dev_file_dir, 防止每次train都重新generate feature, cross-e
l2r_ranker.train('./data/hw2_relevance.train.csv', './data/hw2_relevance.dev.csv
```

```
Train and Dev Data has been loaded from ./cache/rel_train_dev.npz......X_shape:
(17700, 7530), y_shape: (17700,), qgroups_shape: (177,)
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.012025 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3616
[LightGBM] [Info] Number of data points in the train set: 17700, number of used f
eatures: 538
```

6. Analyze the result

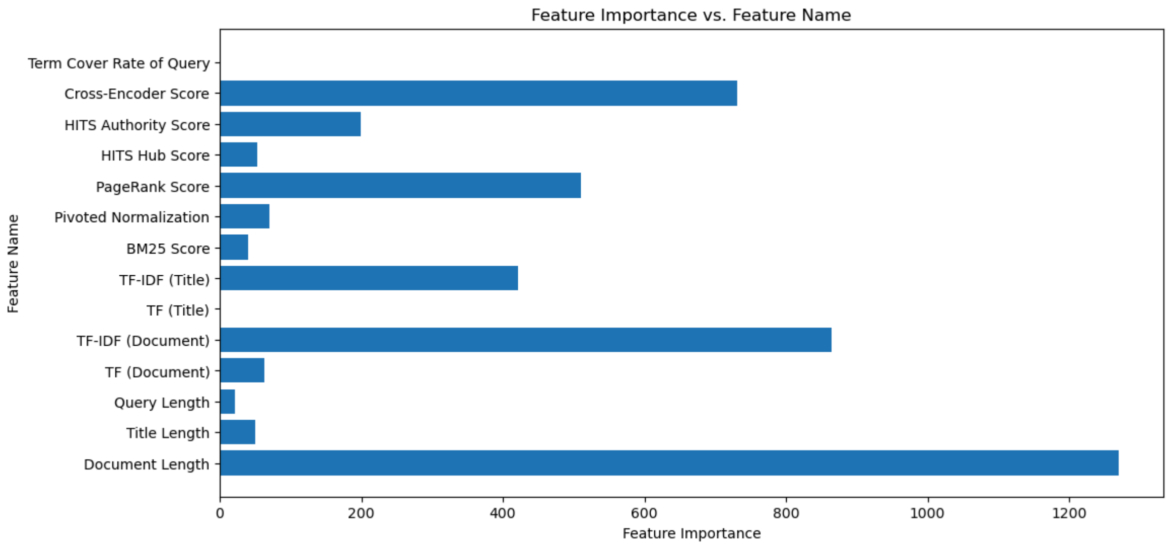
In [9]:
```
eval_results = l2r_ranker.model.model.evals_result_['valid_0']
iterations = range(1, len(eval_results['ndcg@5']) + 1)

plt.figure(figsize=(10, 6))
for key, values in eval_results.items():
    plt.plot(iterations, values, label=key)

plt.title('NDCG Scores Over Dev Dataset')
plt.xlabel('Iterations')
plt.ylabel('NDCG Score')
```

```
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



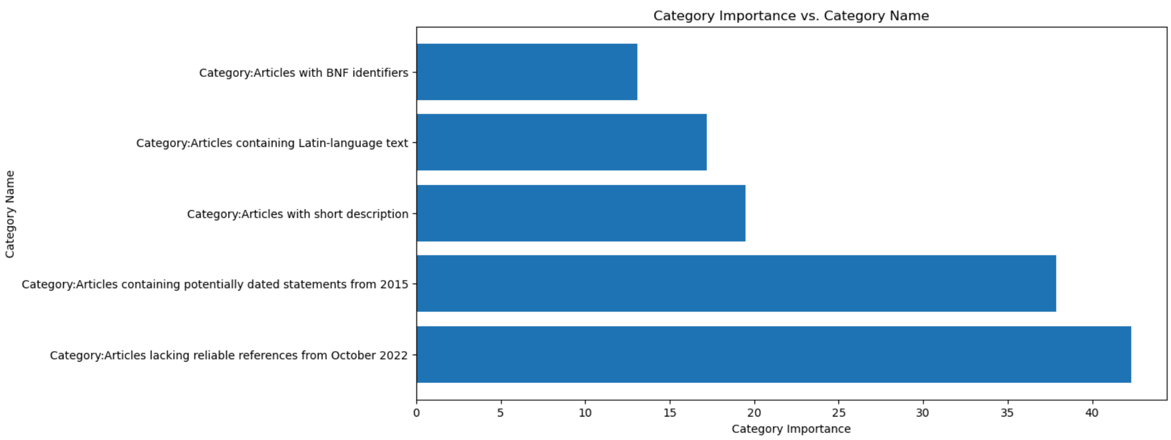
In [18]:

```
feature_import = list(l2r_ranker.model.model.feature_importances_ )
feature_name =[
    "Document Length",
    "Title Length",
    "Query Length",
    "TF (Document)",
    "TF-IDF (Document)",
    "TF (Title)",
    "TF-IDF (Title)",
    "BM25 Score",
    "Pivoted Normalization",
    "PageRank Score",
    "HITS Hub Score",
    "HITS Authority Score",
    "Cross-Encoder Score",
    "Term Cover Rate of Query",
]

# make a plot of feature importance vs. feature name
plt.figure(figsize=(12, 6))
plt.barh(feature_name, feature_import[:14])
plt.xlabel('Feature Importance')
plt.ylabel('Feature Name')
plt.title('Feature Importance vs. Feature Name')
plt.show()
```



In [12]:

```
cat_feature_import = feature_import[14:]
cat_idx_score = []
for i, score in enumerate(cat_feature_import):
    if score > 0:
        cat_idx_score.append((i, score))
cat_idx_score = sorted(cat_idx_score, key=lambda x: x[1], reverse=True)[:5]
cat_names = [l2r_feature_extractor.id_2_recognized_categories[i] for i, _ in cat

plt.figure(figsize=(12, 6))
plt.barh(cat_names, [score for _, score in cat_idx_score])
plt.xlabel('Category Importance')
plt.ylabel('Category Name')
plt.title('Category Importance vs. Category Name')
plt.show()
```



- 7. Evaluate the result: compare simple BM25 with BM25 + Cross-Encoder

In [13]:

```
relevance_path = './data/hw2_relevance.test.csv'
bm25_relevance_test_res = relevance.run_relevance_tests(relevance_path, basic_ra
bm25_relevance_test_res['map'], bm25_relevance_test_res['ndcg']
```

```
100%|██████████| 53/53 [00:34<00:00,  1.53it/s]
100%|██████████| 53/53 [00:00<00:00, 109.60it/s]
```

Out[13]: (0.0656371668164121, 0.37413950616229297)

In [10]:

```
relevance_path = './data/hw2_relevance.test.csv'
lgb_ranker = relevance.run_relevance_tests(relevance_path, l2r_ranker)
```

```
lgb_ranker['map'], lgb_ranker['ndcg']
```

```
100%|██████████| 53/53 [06:41<00:00,  7.57s/it]
100%|██████████| 53/53 [00:00<00:00, 112.49it/s]
```

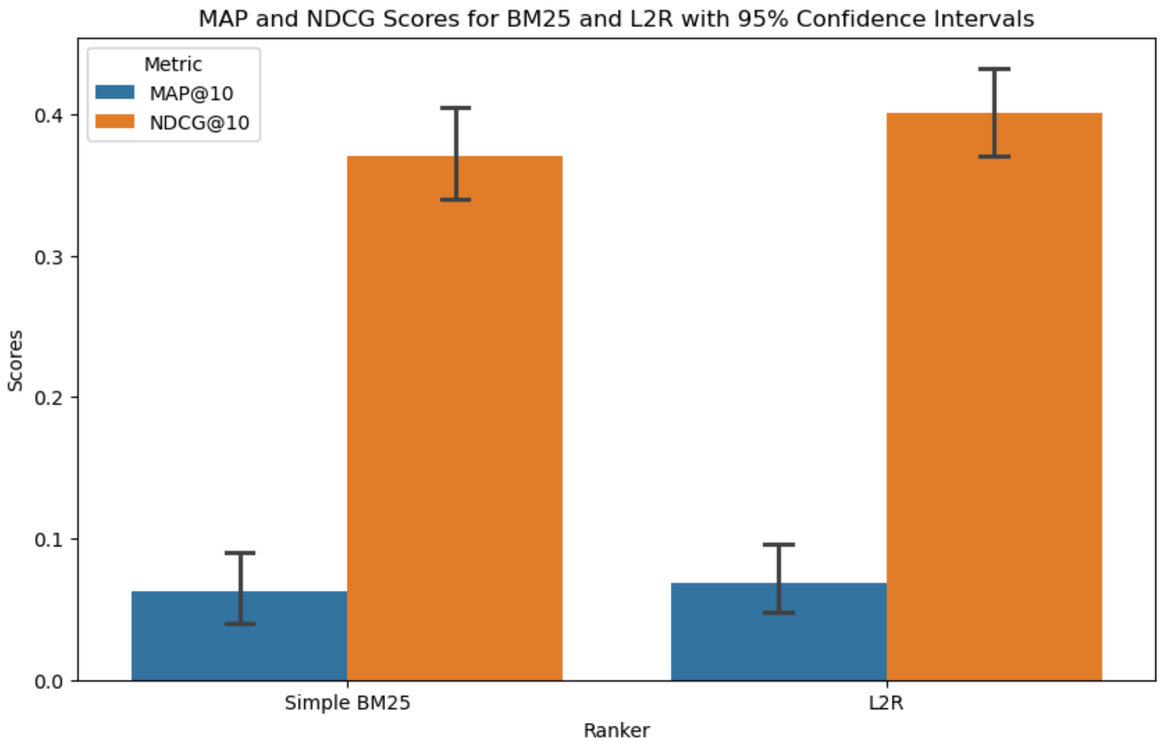
Out[10]: (0.08119347109913146, 0.4128800134923599)

In [10]:
```
# relevance_path = './data/hw2_relevance.dev.csv'
# lgb_ranker_dev = relevance.run_relevance_tests(relevance_path, l2r_ranker)
# lgb_ranker_dev['map'], lgb_ranker_dev['ndcg']
```

```
100%|██████████| 32/32 [04:04<00:00,  7.65s/it]
100%|██████████| 32/32 [00:00<00:00, 115.46it/s]
```

Out[10]: (0.08427579365079364, 0.45818143216517576)

In [15]:
```
bm25_map_scores = bm25_relevance_test_res['map_list']
bm25_ndcg_scores = bm25_relevance_test_res['ndcg_list']
l2r_map_scores = lgb_ranker['map_list']
l2r_ndcg_scores = lgb_ranker['ndcg_list']

# Prepare a DataFrame for plotting
data = pd.DataFrame({
    'Ranker': ['Simple BM25'] * len(bm25_map_scores) + ['Simple BM25'] * len(bm2
              ['L2R'] * len(l2r_map_scores) + ['L2R'] * len(l2r_ndcg_scores),
    'Metric': ['MAP@10'] * len(bm25_map_scores) + ['NDCG@10'] * len(bm25_ndcg_sc
              ['MAP@10'] * len(l2r_map_scores) + ['NDCG@10'] * len(l2r_ndcg_scor
    'Score': bm25_map_scores + bm25_ndcg_scores + l2r_map_scores + l2r_ndcg_scor
})

plt.figure(figsize=(10, 6))
sns.barplot(x='Ranker', y='Score', hue='Metric', data=data, ci=95, capsize=0.1)
plt.xlabel('Ranker')
plt.ylabel('Scores')
plt.title('MAP and NDCG Scores for BM25 and L2R with 95% Confidence Intervals')
plt.show()
```



- Result from HW2
  - map@10 score = 0.0749
  - ndcg@10 score = 0.4074
- Result from HW3
  - map@10 score = 0.0812
  - ndcg@10 score = 0.4128

```
Conclusion: New added feature cross-encoder score contribute both to
map and ndcg performance, which means the introduce of llm help us to
better build the learning to rank system for information retreival.
```