Week 3 Assignment

Exercise 1

With the answers done in the same directory, with `exercise1.py`, here is the most important part of the code where calculation is done.

```
...
def tf_idf(doc: Literal['Doc1', 'Doc2', 'Doc3'], index: Literal['car',
'auto', 'insurance', 'best']) -> float:
    # function implementation
    return df_table1[doc][index] * df_table2['idf'][index]

for doc in ['Doc1', 'Doc2', 'Doc3']:
    for index in ['car', 'auto', 'insurance', 'best']:
        print(f"tf_idf({doc}, {index}) = {tf_idf(doc,index)}")
```

Here is the results for each `td_idf(doc, index)` values.

```
tf_idf("Doc1", "car") = 44.55
tf_idf("Doc1", "auto") = 6.24
tf_idf("Doc1", "insurance") = 0.0
tf_idf("Doc1", "best") = 21.0
tf_idf("Doc2", "car") = 6.6
tf_idf("Doc2", "auto") = 68.64
tf_idf("Doc2", "insurance") = 53.46
tf_idf("Doc2", "best") = 0.0
tf_idf("Doc3", "car") = 39.599999999999994
tf_idf("Doc3", "auto") = 0.0
tf_idf("Doc3", "insurance") = 46.980000000000004
tf_idf("Doc3", "best") = 25.5
```

Exercise 2

With the answers in `exercise2.py`, here's the most important part of the code.

```
...
def ppmi(w: Literal["w1","w2"], c: Literal["c1","c2","c3"]):
    total_sum = df_co_occurence_matrix.values.sum()
    prob_w_c = df_co_occurence_matrix[c][w] / total_sum
    prob_w = df_co_occurence_matrix.loc[w].sum() / total_sum
    prob_c = df_co_occurence_matrix[c].sum() / total_sum

    pmi = log2(prob_w_c / (prob_c * prob_w))
    return max(pmi, 0)
```

```
for w in ["w1", "w2"]:
    for c in ["c1", "c2", "c3"]:
        print(f"ppmi({w}, {c}) = {ppmi(w, c)}")
```

Here is the result, the `ppmi(w,c)` for each is...

```
ppmi(w1, c1) = 0
ppmi(w1, c2) = 0.8300749985576877
ppmi(w1, c3) = 0
ppmi(w2, c1) = 0.12553088208385882
ppmi(w2, c2) = 0
ppmi(w2, c3) = 0.3479233034203066
```

Exercise 3

Important part of the code

```
...
# Cosine similarity
def cosine_similarity(vector1: List[float], vector2: List[float]) ->
float:
    dot_product = sum(a * b for a, b in zip(vector1, vector2))
    magnitude1 = math.sqrt(sum(a * a for a in vector1))
    magnitude2 = math.sqrt(sum(b * b for b in vector2))
    if magnitude1 == 0 or magnitude2 == 0:
        return 0.0
    return dot_product / (magnitude1 * magnitude2)

# Euclidean distance
def euclidean_distance(vector1: List[float], vector2: List[float]) ->
float:
    return math.sqrt(sum((a - b) ** 2 for a, b in zip(vector1, vector2)))

for vector_key in vectors.keys():
    print(f"cosine_similarity(x, {vector_key}) = {cosine_similarity(x,
vectors[vector_key])}")
    print(f"euclidean_distance(x, {vector_key}) = {euclidean_distance(x,
vectors[vector_key])}")
```

Here is the result

```
cosine_similarity(x, a) = 0.8944271909999159
euclidean_distance(x, a) = 1.5811388300841898
cosine_similarity(x, b) = 0.9999999999999998
euclidean_distance(x, b) = 2.8284271247461903
```

```
cosine_similarity(x, c) = 0.9899494936611665
euclidean_distance(x, c) = 7.211102550927978
```

Exercise 4

The code used.

```python
def jaccard_similarity(doc1: str, doc2: str) -> float:
    # Tokenize the documents into sets of words
    set1 = set(doc1.split())
    set2 = set(doc2.split())

    # Compute the intersection and union of the sets
    intersection = set1.intersection(set2)
    union = set1.union(set2)

    # Calculate the Jaccard similarity
    return len(intersection) / len(union)

# Example usage
doc1 = "we love information retrieval course"
doc2 = "information retrieval is a course offered in sutd"

similarity = jaccard_similarity(doc1, doc2)
print(f"Jaccard Similarity: {similarity}")
```

Here is the result

```
Jaccard Similarity: 0.3
```