**Project Report**

**Introduction**: The project aims to create a search engine that indexes a set of documents, gives a set of queries, returns the top relevant documents for each query. The project uses the Inverted Index as the primary data structure for indexing and retrieval. The project implemented various term weighting and normalization schemes and evaluated the system's performance in terms of precision and recall for each query under different query settings.

**Design:** The project's primary data structure is the Inverted Index, which is a mapping between terms and the documents that contain those terms. Each entry in the inverted index contains a list of documents that contain the term and the frequency of the term in each document. The project used the following term weighting and normalization schemes, finding the system performance using precision and Recall.

**Term Frequency (TF) Weighing**: The term frequency weighing scheme assigns a weight proportional to the frequency of the term in the document.

**TF-IDF Weighing: The TF-IDF (Term Frequency**-**Inverse Document Frequency**) weighing scheme assigns a weight that is proportional to the term frequency in the document and inversely proportional to the term's frequency in the entire collection of documents.

**Process for executing the Search Engine:**

This is a simple program written in Python that creates a basic search engine using the vector space model. The computer program looks at different files with information about things like documents, questions, and words that aren't useful for searching. It gets the information ready for searching by organizing it into helpful lists.

In simple terms, preprocessing means breaking up the text into words, taking out commonly used words, shortening the remaining words, and listing each word with how many times it appears in different documents.

The code creates a tool that can read a text file containing a bunch of questions in a particular way. It takes the important parts of each question and separates them into three different groups: the title, description, and the part that tells a story. These three pieces of information are given back as individual strings.

The writing code includes a tool that figures out how much two pieces of text are alike. This means figuring out how important each word is in a search query or document, and then multiplying those weights together.

In the end, there is a set of instructions to search for something. It looks at the words you typed in and compares them to all the things in its collection. The highest matching documents are shown as search results.

In summary, the code creates a simple search engine using a certain method, but it may not work well for really big projects.

**Performance Evaluation:** The project evaluated the system's performance in terms of precision and recall for each query under different query settings. The project used the following query settings:

**Title Query:** The system considers only the main query (title) to retrieve relevant documents.

**Title and Description Query**: The system considers the main query (title) and the description to retrieve relevant documents.

**Title, Description, and Narrative Query:** The system considers the main query (title), the description, and the narrative to retrieve relevant documents.

The project used the following evaluation metrics:

**Precision:** The fraction of retrieved documents that are relevant to the query.

**Recall:** The fraction of relevant documents that are retrieved by the system.