Dissertation on

"Linguistic Analysis of Indo-European Languages"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Roshan U	**01FB15ECS246**
Sanath Bhimsen	**01FB15ECS260**
Mukesh M Karanth	**01FB15ECS361**

Under the guidance of

Internal Guide
Shreekanth M Prabhu
Associate Professor,
PES University

January – May 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Linguistic Analysis of Indo-European Languages'

is a bonafide work carried out by

Roshan U	**01FB15ECS246**
Sanath Bhimsen	**01FB15ECS260**
Mukesh M Karanth	**01FB15ECS361**

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2019 – May. 2019. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature Shreekanth M Prabhu Associate Professor	Signature Dr. Shylaja S S Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the project entitled **Linguistic Analysis of Indo-European Languages** has been carried out by us under the guidance of Prof. Shreekanth M Prabhu and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2019. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

01FB15ECS246	**Roshan U**
01FB15ECS260	**Sanath Bhimsen**
01FB15ECS361	**Mukesh M Karanth**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our guide Prof. Shreekanth M Prabhu for his continuous guidance and insights he provided throughout the last few months. A lot of the resources he shared with us were really helpful going forward in the project.

We would like to express our gratitude towards the project coordinators Ms. Preet Kanwal and Ms. Sangeetha for their constant guidance in every step of our project progress.

We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. K.N.B. Murthy, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way.

Finally, this project could not have been completed without the continual support and encouragement we received from our parents.

ABSTRACT

The goal of this project is to perform linguistic analysis of indo european languages by selecting languages that belong to this family and finding out which is the most central language amongst these languages using Social Network Analysis techniques.

We also compute the similarities between the languages chosen. In our analysis we seek to the show the links between languages as a network rather than the conventional tree structure.

This helps us understand the relationship better at a smaller level of intricacy.

This report contains in it the various references we have used to complete our project, it also contains the scope details, the exploration steps and methodologies we used, the testing approaches and specifications utilized, the outcomes of the project along with the implementation code and finally the conclusion of the project as a whole.

TABLE OF CONTENTS

TABLE OF IMAGES

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Acronyms

- API: Application Program Interface
- GUI: Graphical User Interface
- RAM: Random Access Memory

Abbreviations

- IE: Indo-European
- PIE: Proto Indo-European
- HGT: Horizontal Gene Transfer
- OS: Operating System

CHANGE HISTORY

S.No	Date	Document Version No.	Change Description	Reason of Change
1	25/04/2019	1	Shift to research template	Implemented in regular template
2	26/04/2019	2	Added Results of NodeXL	Project progression

CHAPTER 1: INTRODUCTION

We will start with the introduction to what Indo-European languages are and a brief idea on how they spread and developed across the continents.

The Indo-European languages are a family of languages that consist of several hundred related languages and dialects, including some of the most popularly studied and used languages like Sanskrit, Persian, Russian, French, English, etc.

There are about Four Hundred and Forty Five living Indo-European languages currently known to us, according to the estimate by Ethnologue a language documenting encyclopedia for all the world's known languages, with over two thirds(about three hundred and thirteen)of them belonging to the Indo-Iranian branch of the Indo-European language family tree. The most widely spoken Indo-European languages that are currently spoken by native speakers of that language are Hindustani (Hindi-Urdu sub-tree), Spanish, English, Portuguese, Bengali, Punjabi, and Russian. Each of them have a language user base with over a hundred million speakers. Also, languages like German, French, Marathi, Italian, and Persian also having more than fifty million speakers of the language to this day.

Today, nearly Forty-Two percent of the human population (approximately 3.2 billion) speaks an Indo-European language as their first language, which is by far the highest speaker base of any language family in recorded history.

The Indo-European family includes most of the modern languages of Europe; notable exceptions include Hungarian, Turkish, Finnish, Estonian, Basque, Maltese, and Sami. The Indo-European family is also represented in Asia with the exception of East and Southeast Asia. It was predominant in ancient Anatolia (present-day Turkey), the ancient Tarim Basin (present-day Northwest China) and most of Central Asia until the medieval Turkic and Mongol invasions. Outside Eurasia, Indo-European languages are dominant in the Americas and much of Oceania and Africa, having reached there during the Age of Discovery and later periods. Indo-European languages are also most commonly present as minority languages or second languages in countries where other families are dominant.

With written evidence appearing since the Bronze Age in the form of the Anatolian languages and Mycenaean Greek, the Indo-European family is significant to the field of historical linguistics as possessing the second-longest recorded history, after the Afroasiatic family,

although certain language isolates, such as Sumerian, Elamite, Hurrian, Hattian, and Kassite are recorded earlier.

Theory of Evolution of Indo-European language

All Indo-European languages are descendants of a single prehistoric language, reconstructed as Proto-Indo-European, spoken sometime in the Neolithicera. Although no written records remain, aspects of the culture and religion of the Proto-Indo-Europeans can also be reconstructed from the related cultures of ancient and modern Indo-European speakers who continue to live in areas to where the Proto-Indo-Europeans migrated from their original homeland.Several disputed proposals link Indo-European to other major language families. Although they are written in Semitic Old Assyrian, the Hittite loanwords and names found in the Kültepe texts are the oldest record of any Indo-European language.

During the nineteenth century, the linguistic concept of Indo-European languages was frequently used interchangeably with the racial concepts of Aryan and Japhetite.

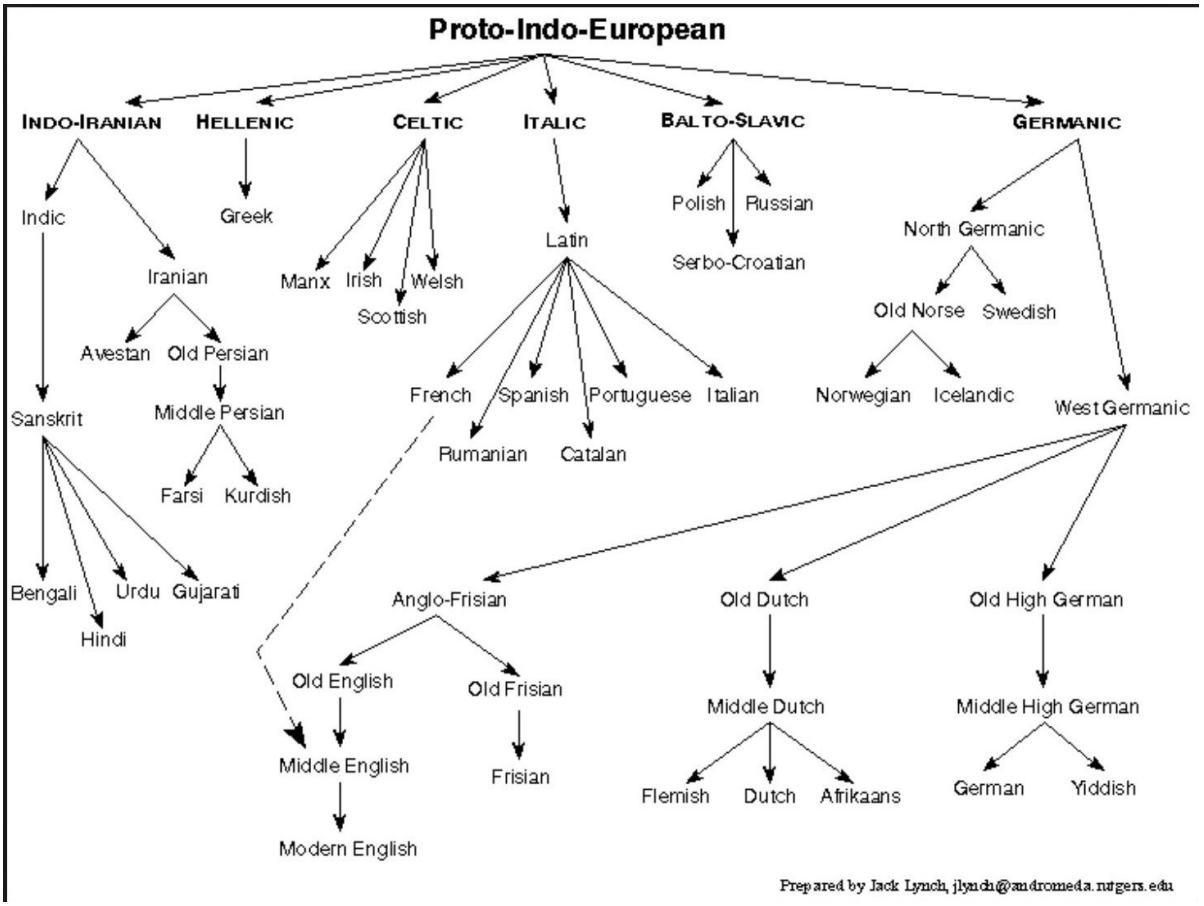

Figure 1: Tree model of evolution of the languages of today

Current Visualization of Evolution of Languages

However, the "tree model" is considered an appropriate representation of the genealogical history of a language family if communities do not remain in contact after their languages have started to diverge. In this case, subgroups defined by shared innovations form a nested pattern. The tree model is not appropriate in cases where languages remain in contact as they diversify; in such cases subgroups may overlap, and the "wave model" is a more accurate representation. Most approaches to Indo-European subgrouping to date have assumed that the tree model is by-and-large valid for Indo-European;however, there is also a long tradition of wave-model approaches.

In addition to genealogical changes, many of the early changes in Indo-European languages can be attributed to language contact. It has been asserted, for example, that many of the more striking features shared by Italic languages (Latin, Oscan, Umbrian, etc.) might well be areal features. More certainly, very similar-looking alterations in the systems of long vowels in the West Germanic languages greatly postdate any possible notion of a proto-language innovation (and cannot readily be regarded as "areal", either, because English and continental West Germanic were not a linguistic area). In a similar vein, there are many similar innovations in Germanic and Balto-Slavic that are far more likely areal features than traceable to a common proto-language, such as the uniform development of a high vowel (**u* in the case of Germanic, **i/u* in the case of Baltic and Slavic) before the Proto-Indo-European syllabic resonants **r̥,* *l̥, *m̥, *n̥*, unique to these two groups among Indo-European languages, which is in agreement with the wave model. The Balkan sprachbund even features areal convergence among members of very different branches.

An extension to the *Ringe-Warnow model of language evolution*, suggests that early Indo-European language had featured limited contact between distinct lineages, with only the Germanic subfamily exhibiting a less treelike behaviour as it acquired some characteristics from neighbours early in its evolution. The internal diversification of especially West Germanic is cited to have been radically non-treelike.

Proto-Indo-European

The proposed Proto-Indo-European language (PIE) is the reconstructed common ancestor of the Indo-European languages, spoken by the Proto-Indo-Europeans. From the 1960s, knowledge of Anatolian became certain enough to establish its relationship to PIE. Using the method of internal reconstruction an earlier stage, called Pre-Proto-Indo-European, has been proposed.

PIE was an inflected language, in which the grammatical relationships between words were signaled through inflectional morphemes (usually endings). The roots of PIE are basic morphemes carrying a lexical meaning. By addition of suffixes, they form stems, and by addition of endings, these form grammatically inflected words (nouns or verbs). The reconstructed Indo-European verb system is complex and, like the noun.

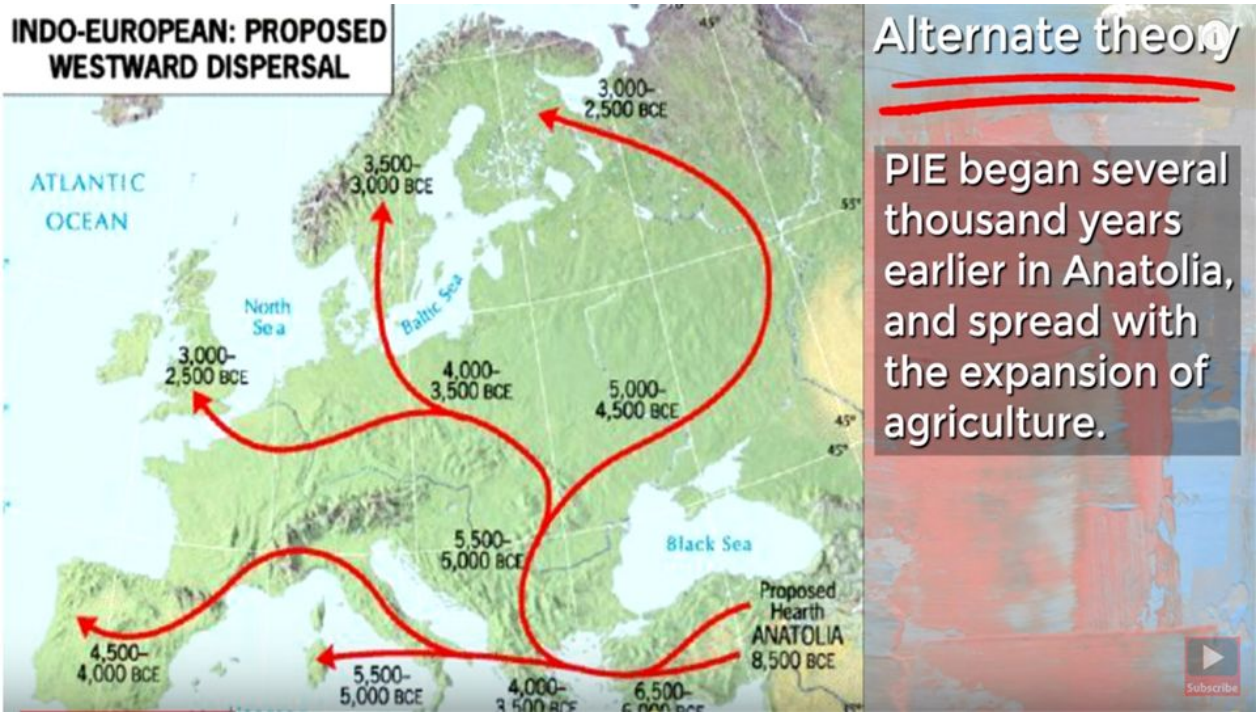

Figure 2: Alternate theory of Proto-Indo-European language evolution and spread

CHAPTER 2: PROBLEM STATEMENT

Based on the provided introduction to the indo-european languages given in the above section and its theory of evolution and spread, the current model of the Indo-European languages is visualised and foretold as a predominant-tree like structure, that is, all languages developed uniquely, individually and independently in their own timelines without the intervention of influence from other languages in the growth of these languages, this implies that all languages developed strictly divergently with little frequency of borrowing.

For us, learning about this fact seemed like the idea of evolution of these languages did not develop from an open minded approach which took all possible considerations for the evolution of these languages from an initial language. Hence we thought that the current model of Indo-European language evolution theory might be a biased model due to the limited considerations and the restricted visualisation of the languages.

Therefore, We want to broaden the considerations taken for the proposed evolution of the Indo-European language family by including possibilities of word transfers that occured between people of various language origins, that may have lead to influence in the development of a new resultant language and mutual growth, which means that people from different linguistic backgrounds may have communicated with each other and imbibed certain words and phrases from each other's language and resulted inevitable in the growth of both the languages involved.

After using these ideas and considerations into the theory of evolution of the languages in Indo-European, we want to come up with a better, more realistic visualization of the Indo-European Languages, one that may resemble an network with various links and crosslinks that connect languages that were thought to have developed independently.

CHAPTER 3: LITERATURE SURVEY

History

In the 16th century, European visitors to the Indian subcontinent began to notice similarities among Indo-Aryan, Iranian, and European languages. In 1583, English Jesuit missionary and Konkani scholar Thomas Stephens wrote a letter from Goa to his brother (not published until the 20th century) in which he noted similarities between Indian languages and Greek and Latin.

Another account was made by Filippo Sassetti, a merchant born in Florence in 1540, who travelled to the Indian subcontinent. Writing in 1585, he noted some word similarities between Sanskrit and Italian (these included *devaḥ*/*dio* "God", *sarpaḥ*/*serpe* "serpent", *sapta*/*sette* "seven", *aṣṭa*/*otto* "eight", and *nava*/*nove* "nine").However, neither Stephens' nor Sassetti's observations led to further scholarly inquiry.

In 1647, Dutch linguist and scholar Marcus Zuerius van Boxhorn noted the similarity among certain Asian and European languages and theorized that they were derived from a primitive common language which he called *Scythian*. He included in his hypothesis Dutch, Albanian, Greek, Latin, Persian, and German, later adding Slavic, Celtic, and Baltic languages. However, Van Boxhorn's suggestions did not become widely known and did not stimulate further research.

Ottoman Turkish traveler Evliya Çelebi visited Vienna in 1665–1666 as part of a diplomatic mission and noted a few similarities between words in German and in Persian. Gaston Coeurdoux and others made observations of the same type. Coeurdoux made a thorough comparison of Sanskrit, Latin and Greek conjugations in the late 1760s to suggest a relationship among them. Meanwhile, Mikhail Lomonosov compared different language groups, including Slavic, Baltic ("Kurlandic"), Iranian ("Medic"), Finnish, Chinese, "Hottentot" (Khoekhoe), and others, noting that related languages (including Latin, Greek, German and Russian) must have separated in antiquity from common ancestors.

Some of the papers we referred are :

1. The Origin of Indo-European Languages

This paper tries to explain the origin, spread and evolution of the so called Indo-European languages by stating how the languages developed and evolved as a result of not only conquests and war but also as a result of peaceful agricultural diffusion. The paper tries to talk about the prehistoric processes that were the underlying factors of the relationships between the languages. It also talks about the traditional view of how the languages spread via nomadic horsemen and about the recent, more modern views on how the languages are thought to have developed.[1]

2. Classification of the Indo-European languages using a phylogenetic network approach

This paper talks about how the words from Indo-European languages can be used to compute the hidden links and ties amongst the languages using similarity measures like edit distances etc. to convert the predominant tree like evolution structure into a more network like structure with cross linkages.

Discovering the origin of the Indo-European (IE) language family is one of the most intensively studied problems in historical linguistics. Gray and Atkinson (2003) inferred a phylogenetic tree (i.e., additive tree or X-tree, Barthelémy and Guénoche 1991) of the IE family, using bayesian inference and rate-smoothing algorithms, based on the 87 Indo-European language data set collected by Dyen et al. (1997). When conducting their classification study, Gray and Atkinson assumed that the evolution of languages was strictly divergent and the frequency of word borrowing (i.e., horizontal transmission of individual words) was very low. As consequence, their results suggested a predominantly tree-like pattern of the IE language evolution. In our opinion, only a network model can adequately represent the evolution of the IE languages. We propose to apply a method of horizontal gene transfer (HGT) detection (Makarenkov et al. 2006) to reconstruct a phylogenetic network depicting the evolution of the IE language family.[2]

3.Evolving a Framework to interpret the Vedas

The Vedas are considered to be the oldest surviving literature of mankind. They are preserved in an intact form through memorizing and transmission from generation to generation over centuries, thus very likely to have an unaltered, authentic record. Vedas have been extensively studied by large number of scholars from diverse disciplines for the last two centuries. They abound in poetic metaphor and allusions yet give an account of people, places and events along with the prevailing belief systems at that time. The belief systems in Vedas have commonality with Iranian and European traditions, collectively referred to as Indo-European. Many scholars have studied Vedas to find the footprints and trail of Indo-Europeans whose languages have noticeable commonality.

In this paper we make an attempt to evolve models and frameworks to interpret Vedas, in particular references to Vedic Gods in the Rig Veda. We believe the analysis of Vedas can throw up useful frameworks and methods to understand present day social information which is generally spread across space, time and contexts.[3]

4. Clustering Semantically Equivalent Words into Cognate Sets in Multilingual Lists

Word lists have become available for most of the world's languages, but only a small fraction of such lists contain cognate information. We present a machine-learning approach that automatically clusters words in multilingual word lists into cognate sets. Our method incorporates a number of diverse word similarity measures and features that encode the degree of affinity between pairs of languages. The output of the classification algorithm is then used to generate cognate groups. The results of the experiments on word lists representing several language families demonstrate the utility of the proposed approach. [4]

5. Mapping the Origins and Expansion of the Indo-European Language Family Remco

There are two competing hypotheses for the origin of the Indo-European language family. The conventional view places the homeland in the Pontic steppes about 6000 years ago. An alternative hypothesis claims that the languages spread from Anatolia with the expansion of farming 8000 to 9500 years ago. We used Bayesian phylogeographic approaches, together with basic vocabulary data from 103 ancient and contemporary Indo-European languages, to explicitly model the expansion of the family and test these hypotheses. We found decisive support for an Anatolian origin over a steppe origin. Both the inferred timing and root location of the Indo-European language trees fit with an agricultural expansion from Anatolia beginning 8000 to 9500 years ago. These results highlight the critical role that phylogeographic inference can play in resolving debates about human prehistory. [5]

CHAPTER 4: PROJECT SCOPE

Scope

The scope of the project is subject to the project being a minor project having a 8 credit weightage and it is to be developed with heavy time constraints, that is, project work will be done only on the weekends because all of the team members are interns who work during the weekdays.

Hence we are going to be scaling down the size and extent of this research project by means of certain limited considerations like:

1. Making use of transliterated words along with manually phonetically translated words in case of any missing data.
2. The number of words will be limited to a maximum of three hundred words that consist of stop words, nouns, verbs, adjectives and adverbs.
3. We will only be using a select number of centrality and similarity measures like Closeness Centrality, Between-ness Centrality and similarity measures like Levenshtein Edit distance, cosine similarity for the computations.

Outcomes

The outcome of the project will be considered fruitful or successful after we obtain a visualization for all the considered languages at hand, for their similarities and ties between each other; represented in the form of a network with distinct central language and words that link other languages together.

Requirements

1. Hardware Requirements
    a. Processor: Intel-core i3 and above
    b. RAM: 8GB and above
    c. Memory: 1-2 GB
    d. Internet Speed: 50Mbps or higher

2. Software Requirements

This section shall specify the use of other required software products (e.g. a data management system, an operating system, or a mathematical package), and interface with other application systems (e.g. the linkage between an accounts receivable system and a general ledger system).

a. Operating System: Windows 10
b. Data Management System: Windows Excel
c. Programing Languages:
    - Python
    - R
d. Packages:
    - iGraph – R
    - scipy
    - numpy
    - nltk
    - pandas
    - math
    - difflib
    - lingpy
    - googletrans
e. Applications
    - Windows Office 365
    - R Studio
    - Jupyter Notebook
    - NodeXL

3. Data Requirements

The data being used is a collective dataset that contains around 200+ words in English which comprise of stop words, nouns and verbs which are then transliterated into the other languages that we are using like French, Sanskrit, etc to use as our core database for evaluations and analytics. The words in English themselves have been obtained from ‘Langinfo’ website, and mostly with the help of Google Dictionary. The transliterated words have been solely obtained via Google Translate API.

Methodology

- **Proposed Approach**
  - Collect the dataset of Indo-European languages from Langfocus website and other similar websites.
  - Select a few key languages and pre-process the dataset for any discrepancies.
  - Perform analysis on the dataset, by getting distance between languages by the closeness of their words using distance measures like Levenshtein distance, etc. and centrality measures.
  - Understand and apply Horizontal Gene Transfer Detection Algorithm.
  - Combine all the results and visualise the dataset to obtain a new and better model of the layout of Indo-European Languages.
- **High Level System Architecture**
  - Dataset: This module is meant to house all the words of the various languages we have considered so far and it is also meant to be a live dataset that adapts and evolves over time based on the requirements of the Customer.
  - Analytical Backend: This module takes the words from the dataset as input and performs various analytics on the data such as similarity and closeness measures, clustering, etc. It then gives the output as a table to the next module, this makes the module an intermediate.
  - Visualization: This module takes the output of the analytical module in the form of a table and then plots the data points as nodes in the graph and connects the nodes based on their similarity with one another to obtain a network like structure to help get a better understanding of the Languages and their connections with one another.

Demonstration of Outcome

This section shall specify how you propose to demonstrate the outcomes of your project.

- Will there be a GUI – No, currently GUI is not part of the project at hand due to its research oriented objectives. Hence this project will be a Console based interface.
- Will there be analysis of data? - Yes, The analyzed data will be interpreted as graphical visualizations and normal visualizations using python and R together.
- Will there be proof of a theorem? – Yes, we are trying to tackle the current model that states the Evolution of Indo-European language is a pre-dominant tree like structure. We will try to interpret the evolution of IE language as a network with concepts like mutual growth and similarities.

CHAPTER 5: INTERIM EXPLORATION

Chapters

(1) Data Collection:

(a) After obtaining a list of all required English words, it was saved in excel and more columns for the other languages were created as null attributes.

(b) A python script was then created and in it google translate api was called for each of the words in the English column

(c) For each translation, its pronunciation was checked and appended if it exited, otherwise the vanilla translated word was appended to the respective column to which the word belonged to.

(2) Pre-Processing:

(a) Once the dataset was created, each word was replaced with its phonetic pronunciation using manual or automatic conversion wherever applicable.

(b) The Persian language was deleted and in its place Russian was added due to translation errors.

(3) Similarity Computations:

(a) A row from the processed dataset was taken and then converted into a word matrix

(b) Then, the edit distance between every two words in the matrix was computed and stored in the matrix

(c) Finally, a link was established between two words if their similarity exceeded a particular threshold

(d) The threshold was determined by exhaustive exploration.

(e) The results were put into a new dataset that had the word, its parent language and its cluster number

(4) Visualization:

(a) The resultant dataset from the computations were fed into a dataframe and was converted into nodes of language and cluster.

(b) The nodes were then converted into a network and the underlying communities were highlighted to show the result of our computations.

CHAPTER 6: TEST STRATEGY

- **Introduction**

  This document is aimed at formally documenting and reporting the testing process that was used during the development of the project. It helps in recording the various methods, approaches and test cases used for the purpose of testing. It also has the entire plan of action and the roles assigned to each member of the team.

  The purpose of this document is keep a record of all the processes and methods used along with the establishment of proof for the responsibilities delegated to each member of the team.

  Scope

  The Testing process for the project being developed is aimed for test the project at each and every step during the development cycle and to test the whole product after integration. Hence, Testing is a part of every phase of the product development.

- **Test Strategies**

  This section shall clearly define the test strategies that are planned for testing the product. The Test strategy may be for the following types of tests: -

Stress	User Friendliness	Function
Error Recovery	Error Handling	Boundary
Disaster recovery	Concurrent Testing	

  - User Friendliness: Check the simplicity of the visualization in terms of readability by adjusting factors like size, color, font, layout, etc.
  - Function: At each stage of the project execution, check if the output is saved correctly, also put in precautionary and verbose error statements to be printed if the code fails for some reason.
  - Error Handling: Throw error number and verbose explanation for an expected error which occurs. Safely terminate the program and do not save outputs in case of unexpected errors.

- **Performance Criteria**

  The performance criteria for the testing process was the correctness of the output rather than the time taken or efficiency of the developed project.

  This makes sure the output is valid for any data, hence helping achieve the goal of getting a new visualization of the Indo-European language evolution theory.

  The performance criteria was checked in accordance to the confusion matrix evaluation methodology.

- **Test Environment**

  System configurations

  - Hardware:
    - Processor: Intel Core i7 Octa-Core
    - RAM: 8GB
    - Memory: 2TB HDD + 256GB SSD
    - Graphics: Nvidia GTX 960M
  - Software:
    - System OS: Windows 10
    - Software applications:
      - Jupyter Notebook
      - R Studio
      - Microsoft Excel

- **Risk Identification and Contingency Planning**

This section identifies the risks that are associated during the testing

Risk #	Risk	Nature of Impact	Contingency Plan
1	GoogleAPI shutdown	Medium	**Restart the API after a time period**
2	No transliteration in google API	High	**Manually search online for a transliteration with correct phonetics**

- **Roles and Responsibilities**
  - Roshan U:
    - Roles: Manual testing of dataset
    - Responsibilities:
      - To ensure the transliterated word is present in the correct language
      - Ensure that the phonetically transliterated word is valid
      - Verification of dataset collection.
  - Sanath Bhimsen:
    - Roles: Unit Testing
    - Responsibilities:
      - Test whether the code developed for creating dataset works correctly
      - Test whether the code developed for measuring similarity between languages works correctly
      - Test the correct storage of results
  - Mukesh M Karanth:

- Roles: Integrity Testing
- Responsibilities:
  - Test the overall smooth functioning of the whole product
  - Test the correctness of the visualization code
  - Test the graphical outputs individually

- **Test Schedule**

  Unit Testing:

  - Dataset testing: 2-4th April 2019
  - Code testing: 3-4th April 2019
  - Visualization testing: 4-5th April 2019
  - Integrity testing: 6th April 2019

- **Test Tools Used**

  Since the project is an exploratory research oriented project, we did not make use of any Testing tools, we resorted to domain knowledge verification and brute force validation methods.

- **Acceptance Criteria**

  - Dataset:
    - Words present in dataset must be similar to the phonetic pronunciation of the transliterated word
  - Code:
    - The code must be able to give at least 2 or more clusters as output after similarity analysis of the words
    - It must be able to assign each word with a cluster number
  - Visualization:
    - The Graphical visualization must exactly portray the analytical results
    - The visualization must be able to capture the hidden community structure from the network.

- **Test Case List**

  This section shall clearly define the test cases that are planned for testing. The following information shall be mentioned: -

Test Case Number	Test Case	Required Output
1	Transliteration of English words	Valid transliterated words from google translate.
2	Clustering of words after similarity analysis	Formation and association of words into clusters, at least 2-3 clusters.
3	Visualization Readability	Well spread out nodes present within highlighted clusters.

- **Test Data**

  The data used in the testing process was a set of ten to fifteen random words from the actual dataset. This helped in checking the functionality of the code and visualization using the domain knowledge.

  The test data consisted of a set of words which were transliterations of an English word, they were tested for similarity and graphical visualization correctness.

- **Traceability Matrix**

CRS/HLD/LLD Reference Section No. and Name	Test Case Number
Scope - Abstract	2
Scope - Proposed solution	1
Interim Exploration – Exploratory analysis	3

CHAPTER 7: MODULES

1) **Collections of English Words**: In this module, We started off by collecting around 1000 english words and then filtered it down to 300 words suitable words for our study. We made sure that the data had a good number of stop words, nouns, proverbs, adjectives and words representing relationships like Father, Mother, Etc.

2) **Translation of English Words**: Once this was done , The googletrans python API was used in order to translate each english words into the other Indo-European Languages we had chosen. They are German , Italian , French , Spanish , Hindi ,Russian, Latin and Sanskrit.
   Since there is no support for Sanskrit on Google Translate , We used an English to Sanskrit Dictionary in order to find the Sanskrit Representation of the word.

3) **Loading all these words into a dataframe**: The dataset is prepared this way and it is stored in csv format columns represent the languages and we have the words and their representation on each row.

4) **Preprocessing** : Once the dataset was created, each word was replaced with its phonetic pronunciation using manual or automatic conversion wherever applicable.
   As the API might sometimes return the word in its native representation, this would negatively affect our results because the special characters present as a translation of certain words in other languages will not be considered close to a word even if it sounds the same when pronounced, this is because of the way the similarity measure is computed.

5) **Weighted Similarity Graph Using NetworkX** : The python library networkX was used to prepare a visualization that shows how close every pair of languages are. We used weighted edges for the same. In this module, each word of every language taken for the study for a particular meaning will be taken and compared with each other and their similarity score will be computed for every possible pair and the result will be stored and returned as a matrix.

6) **Creation of Distance Matrix**: For every english word we construct distance matrix showing the distance between that word and its representation in all other languages. Since we have nine languages in total we will create a 9x9 matrix that stores the distance matrix for all language's words for a particular meaning. This will be used for the detection of clusters, communities and hidden linkages between the words from different languages. This will give us a basic idea of the potential relationship that might exist between two previously independent languages.

7) **Cognate Cluster Creation**: The distance matrix for each word is passed to the lingpy cognate cluster creation method and it returns a dictionary. The keys of the dictionary represent the cluster number and the value is a list which contains the word that falls in that cluster. These results are stored in a csv file and passed as an input for the visualization modules.

8) **Network Nodes Creation** : The cognate clusters stored in the csv file are read into a dataframe variable in R and are converted into a format readable by the network node creator function. That is, the cluster number and the fused attribute of word and language are passed as the two parameters for the node creation function, the function then creates a base network layout of all the nodes.

9) **Visualization in R**: The network layout is then passed to the community detector which portrays the nodes as individually highlighted clusters of different colors, which give us a good idea of the hidden relationships that may potentially exist between two independent languages.

CHAPTER 8: IMPLEMENTATION

```
!pip install googletrans
```

```
In [1]: import pandas as pd
        import os
```

```
In [13]: from googletrans import Translator
         translator = Translator()
```

The necessary libraries are imported and google trans constructor is set up

```
words_df = pd.read_csv("words.csv")
eng = list(words_df['English'])
eng_words = eng[0:300]
french = []
hindi = []
latin = []
spanish = []
persian = []
german = []
italian = []
russian = []
```

The dataset is read into a pandas dataframe and all the other lists are initialised in order to store the transliterated words.

We will considering 300 words from the english language and this will be passed to googletrans for translation.

```
#English to German

for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='German')
        if(tran_word.pronunciation==None):
            german.append(tran_word.text)
        else:
            german.append(tran_word.pronunciation)
```

For every english word the translation is done to german. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the german list , otherwise we append the text to the list and it is manually checked for pronunciation.

```
#English to French

for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='French')
        if(tran_word.pronunciation==None):
            french.append(tran_word.text)
        else:
            french.append(tran_word.pronunciation)
```

For every english word the translation is done to French. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the French list , otherwise we append the text to the list and it is manually checked for pronunciation.

```
#English to Italian

for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='italian')
        if(tran_word.pronunciation==None):
            italian.append(tran_word.text)
        else:
            italian.append(tran_word.pronunciation)
```

For every english word the translation is done to Italian. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the Italian list , otherwise we append the text to the list and it is manually checked for pronunciation.

```
#English to Hindi

for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='Hindi')
        if(tran_word.pronunciation==None):
            hindi.append(tran_word.text)
        else:
            hindi.append(tran_word.pronunciation)
```

For every english word the translation is done to Hindi. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the Hindi List , otherwise we append the text to the list and it is manually checked for pronunciation.

```
for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='russian')
        if(tran_word.pronunciation==None):
            russian.append(tran_word.text)
        else:
            russian.append(tran_word.pronunciation)
```

For every english word the translation is done to Russian. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the Russian List , otherwise we append the text to the list and it is manually checked for pronunciation.

```
#English to Latin

for word in eng_words:
        print(word)
        translator = Translator()
        tran_word = translator.translate(word , src='en' ,dest='Latin')
        if(tran_word.pronunciation==None):
            latin.append(tran_word.text)
        else:
            latin.append(tran_word.pronunciation)
```

For every english word the translation is done to Latin. The tran_word contains the text and the pronunciation. Since we are interested in the pronunciation of the word if it exists we append it to the Latin List , otherwise we append the text to the list and it is manually checked for pronunciation.

Dataset

English	French	German	Hindi	Italian	Latin	Russian	Sanskrit	Spanish
the	la	das		ila	quod			la
of	de	fon	ka	di	autem	iz		de
to	à	zu	seva mere	aa	ut	vey		a
and	et	unt	tatha	a	et	a takzhe	tu	y
a	une	ein	e	un	autem			uno
in	dans	im	mein	nela	apud	vey		en
is	est	is	hai	eh	is	yavlyayetsya		es
it	il	est	yaha	isso	it	Eto		eso
you	vous	zie	aap	tu	vos	vy	bhavantham	thu
that	cette	das	us	quello	quod	tot	yath	ese
he	il	er	vah	lui	quod	on		el
was	était	waar	tha	era	was	Bylo		estaba
for	pour	zum	ke liye	pera	quia	za	hi	para
on	sur	auf	par	sopra	in	na		en
are	sont	zind	kar rahe hain	siamo	sunt	yavlyayutsya		son
with	avec	mit	saath mein	con	apud	sa	sardham	con
as	comme	wie	jaisa	come	quod	kak		como
I	je	ish	main	io	ego	ya		yo
his	le sien	seina	usake	il suo	eius	yego		su
they	ils	zie	ve	essi	quod	Oni	tah	elyos

Figure 3: Dataset - the collection of words from all 10 languages taken into consideration.

The Dataset was prepared using the words from the english language and the translated words and it was stored in a csv file.

Preliminary Similarity checks

```
import matplotlib.pyplot as plt
import networkx as nx
```

```
G = nx.Graph() #Create a graph object called G
node_list = ['English','German','Italian','Hindi','French','Latin','Spanish']

for node in node_list:
    G.add_node(node)
```

We imported 2 libraries networkx and matplotlib.pyplot. NetworkX is a python library which is most commonly used for visualizing networks and computing network metrics.

We also instantiate the networkX graph and instantiate the possible nodes in the graph.

```
pos=nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos,node_color='skyblue',node_size=2500)
```

```
<matplotlib.collections.PathCollection at 0x1a36a39bda0>
```

```
labels = {}
for node_name in node_list:
    labels[str(node_name)] =str(node_name)

nx.draw_networkx_labels(G,pos,labels,font_size=11)
```

```
{'English': <matplotlib.text.Text at 0x1a36a3ab390>,
 'French': <matplotlib.text.Text at 0x1a36a3a5358>,
 'German': <matplotlib.text.Text at 0x1a36a3a1a20>,
 'Hindi': <matplotlib.text.Text at 0x1a36a3a5e10>,
 'Italian': <matplotlib.text.Text at 0x1a36a3a5860>,
 'Latin': <matplotlib.text.Text at 0x1a36a3ab8d0>,
 'Spanish': <matplotlib.text.Text at 0x1a36a3a1e80>}
```

In this code snippet we choose the layout for a network and the chosen layout is the spring layout. The network labels are also set up and as a result we can see the names of a language along with a particular node.

```
G.add_edge(node_list[0],node_list[1],weight=eng_ger*10)
G.add_edge(node_list[0],node_list[2],weight=eng_ita*10)
G.add_edge(node_list[0],node_list[3],weight=eng_fre*10)
G.add_edge(node_list[0],node_list[4],weight=eng_hin*10)
G.add_edge(node_list[0],node_list[5],weight=eng_lat*10)
G.add_edge(node_list[0],node_list[6],weight=eng_spa*10)

G.add_edge(node_list[1],node_list[2],weight=ger_ita*10)
G.add_edge(node_list[1],node_list[3],weight=ger_fre*10)
G.add_edge(node_list[1],node_list[4],weight=ger_hin*10)
G.add_edge(node_list[1],node_list[5],weight=ger_lat*10)
G.add_edge(node_list[1],node_list[6],weight=ger_spa*10)

G.add_edge(node_list[2],node_list[3],weight=hin_ita*10)
G.add_edge(node_list[2],node_list[4],weight=ita_fre*10)
G.add_edge(node_list[2],node_list[5],weight=ita_lat*10)
G.add_edge(node_list[2],node_list[6],weight=ita_spa*10)

G.add_edge(node_list[3],node_list[4],weight=hin_fre*10)
G.add_edge(node_list[3],node_list[5],weight=hin_lat*10)
G.add_edge(node_list[3],node_list[6],weight=hin_spa*10)

G.add_edge(node_list[4],node_list[5],weight=lat_fre*10)
G.add_edge(node_list[4],node_list[6],weight=fre_spa*10)


G.add_edge(node_list[5],node_list[6],weight=lat_spa*10)
```

Here the edges between every language node is initialized and the thickness of each edge is proportional to how similar the two languages which are connected by the edge are.

```
for weight in unique_weights:
        #4 d. Form a filtered list with just the weight you want to draw
    weighted_edges = [(node1,node2) for (node1,node2,edge_attr) in G.edges(data=True) if edge_attr['weight']==weight]
        #4 e. I think multiplying by [num_nodes/sum(all_weights)] makes the graphs edges look cleaner
    width = weight*len(node_list)*7.0/sum(all_weights)
    nx.draw_networkx_edges(G,pos,edgelist=weighted_edges,width=width)
```

Here the final graph is created accordingly and the matplotlib library will be used to form the visualization.

Cognate Cluster Analysis

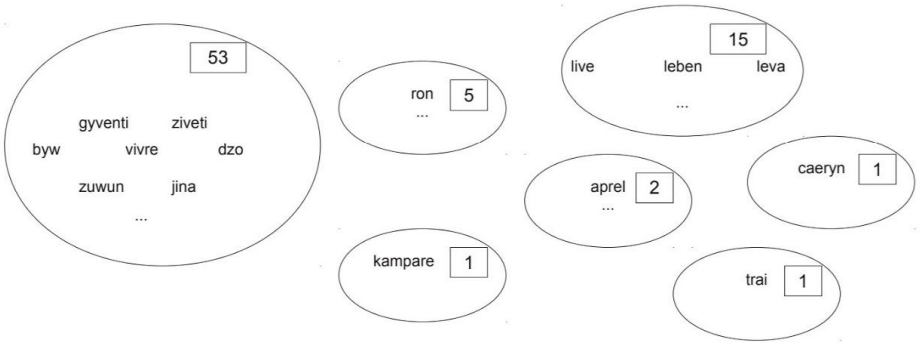

Figure 4: Cognate Clusters example

The above image shows an example of a group of cognate clusters for the word 'live' in english.

Words that are similar tend to be in one cluster. We shall try to emulate this in python on our dataset.

```
import os
import pandas as pd
import numpy as np
from lingpy import *
from lingpy.algorithm import squareform
```

Firstly we import all the required libraries like os and pandas. We also include a python library called Lingpy which is used for linguistic analysis of languages in python.

It enables us to make the cognate clusters.

Before we get into the further details , here is a bit about the python library lingpy:

LingPy is a suite of open-source Python modules for sequence comparison, distance analyses, data operations and visualization methods in quantitative historical linguistics.
The main idea of LingPy is to provide a software package which, on the one hand, integrates different methods for data analysis in quantitative historical linguistics within a single framework, and, on the other hand, serves as an interface for the preparation and analysis of linguistic data using biological software packages.

Lingpy is mainly used for these purposes:

- tokenize and analyze IPA-encoded sequences,
- carry out pairwise and multiple alignment analyses,
- search automatically for cognates across multiple languages,
- calculate lexicostatistic distances between languages,
- reconstruct language phylogenies using basic cluster algorithms, and
- export the results of these analyses to various formats which can be either used as input for external programs, or to visualize the results

```
for k in range(0,300):
    List1 = Target
    List2 = Target

    Matrix = np.zeros((len(List1),len(List2)))
    final_Arr = []
    for i in range(0,len(List1)):
    #temp = []
        for j in range(0,len(List2)):
            Matrix[i,j]=1-difflib.SequenceMatcher(None,df[List1[i]][k],df[List2[j]][k]).ratio()
    #final_Arr.append(temp)

    a = Matrix.tolist()
    cluster = link_clustering(0.2,a,Target)


    c ={}

    for i in cluster:
        temp = []
        a = cluster.get(i)
        for lang in a:
            temp.append(df[lang][k])

        c[i]=temp

    d.append(c)
```

.

In the above code a distance matrix is constructed for every word in english , hence we are talking about a 9 by 9 matrix as we have 9 languages in total.

The distances between the words are computed by the difflib library and feeded into the matrix.

The matrix is provided to lingpy cognate clustering method along with a threshold , any distance value greater than the threshold means the words will not be in the same cluster. This is applied across all words and we get the cognate clusters in the form of a dictionary.

An example is illustrated on the next page.

```
{1: ['padhre', 'padhre'],
 2: ['father', 'fater', 'pater'],
 3: ['pita'],
 4: ['père'],
 5: ['otets'],
 6: ['janaka']}
```

Figure 5: Cognate Cluster List

In the above image the cognate clusters for the word ‘Father’ are shown.

Cluster 1 : ‘Padhre’ means Father in both Spanish and German

Cluster 2 :’Father’ in English , ‘Fater’ in German and ‘Pater’ in Latin fall in one cluster as they are quite similar to each other than they are with the rest.

Cluster 3: ‘Pita’ means Father in Hindi.

Cluster 4 :’Pere’ means Father in French.

Cluster 5:’Otets’ means Father in Russian

Chapter 6:’Janaka’ means Father in Sanskrit.

Before we go further, a brief introduction to the iGraph visualization package in R:

iGraph is an open source package available in R which primarily serves as a library which provides multiple network visualization tools that provide ease of usage along with portability and efficiency.

```
## Download and install the package
install.packages("igraph")

## Load package
library(igraph)
```

Visualization Code in R:

```
# Social Network Analysis using R
#         -Visualisation of clusters for "Father"
library(igraph)
#Download the Language dataset and chose it.
data <- read.csv(file.choose(), header=T)
#Data frame of cluster number and combined attributes language and word
y <- data.frame(data$cluster, paste(data$word, data$Language))
#Data frame of attributes language and word
#y <- data.frame(data$Language, data$word)
#Data frame of cluster number and word
#y <- data.frame(data$cluster, data$word)

#creation of network
net <- graph.data.frame(y, directed=F)

#Community Detection
net <- graph.data.frame(y, directed = F)
cnet <- cluster_edge_betweenness(net)
#plotting the community structure
plot(cnet,
     net,
     vertex.size = 25,
     vertex.label.cex = 1)

```

The above mentioned code enables us to read the cognate cluster file as a dataframe, modify it to suit the network node property by fusing certain attributes, and then feeding this modified dataframe into a network framework. Finally, the network framework variable is plotted as a community network that highlights the communities present in the cognate cluster.

Output visualizations

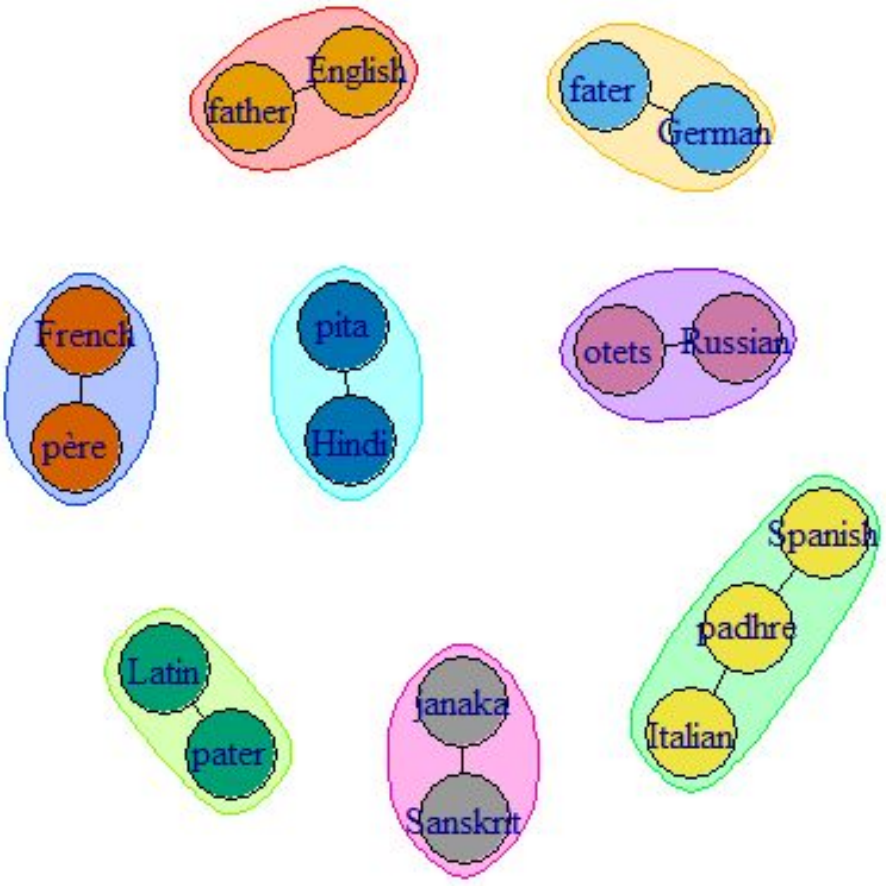

Figure 6: Basic word clustering

This output represents the transliterations of the word ‘Father’ in all the languages, mapped to the language the transliterated word belongs to.

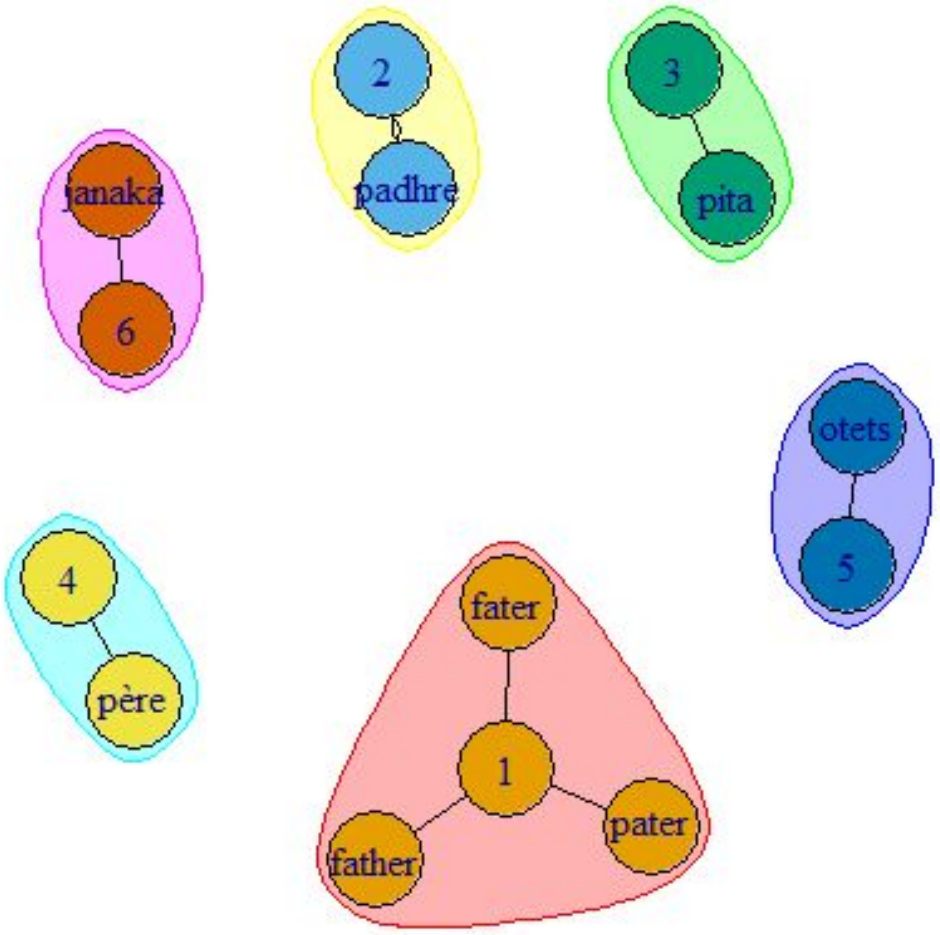

Figure 7: Cognate Clustering Visualization - 'Father'

This figure depicts the word cognate clusters for the word 'Father', it highlights the various clusters are formed after the computation of their similarity measures.

New Approach to Create Database

In our earlier approach we initially collected the english words and once this was done it was translated to the words in the other languages.

This may not be the best approach as the chosen english word may not phonetically be a good match to the translated word.

So what we intend to do is utilize the nltk.wordnet library to find all the synonyms of the english word and then check how similar the synonyms are to the translated word.

```
In [6]: for synset in wordnet.synsets('rejoice'):
            for lemma in synset.lemmas():
                syns.append(lemma.name())

        print(set(syns))

{'wallow', 'triumph', 'joy', 'rejoice', 'exuberate', 'jubilate', 'exult'}
```

The snippet shows the synonyms for the word ‘Rejoice’.

```
{'activate': ['trigger',
  'set off',
  'touch off',
  'actuate',
  'spark off',
  'spark',
  'activate',
  'trigger off',
  'aerate',
  'trip'],
```

Figure 8: Synonyms for the word activate

For every english word , we find the synonyms. We check how close each synonym is the translated word in the other language. If a original english word is closer to most of the translated words then it is retained in our dataset.

If that is not the case and a synonym happens to be closest more number of times , it replaces the original word.

```
for i in range(len(eng_words2)):
    wx = dict()
    mx = -1
    seq = difflib.SequenceMatcher(None,german[i],eng_words2[i])
    scr = seq.ratio()
    syn_list = sd.get(eng_words2[i])
    for word in syn_list:
        s = difflib.SequenceMatcher(None,german[i],word)
        flag = 0
        if(s.ratio()>scr):
            flag = 1
            if(s.ratio()>mx):
                mx = s.ratio()
                w = word

    if(flag==1):
        print('The synonym', w ,'is closer to',german[i],'than', eng_words2[i])
        wx[w] = 1

    else:
        print(eng_words2[i],'the original word is closest to',german[i])
        wx[eng_words2[i]] = 1
        count = count + 1

    cd.append(wx)

    print('\n')
```

The code shows the process of finding synonyms of each english word and comparing it with the translated word from the german language.

```
Out[65]: [{'activate': 7},
          {'annoy': 3, 'bother': 1, 'gravel': 1, 'nark': 1, 'vex': 1},
          {'appreciate': 7},
          {'ascertain': 5, 'control': 1, 'insure': 1},
          {'attain': 7},
          {'bang': 5, 'clap': 1, 'eruption': 1},
          {'belong': 6, 'go': 1},
          {'bounce': 5, 'resile': 1, 'reverberate': 1},
          {'bubble': 7},
          {'conceptualize': 7},
          {'confound': 1, 'confuse': 5, 'discombobulate': 1},
          {'coordinate': 7},
          {'cough': 7},
          {'critique': 6, 'review': 1},
          {'dam': 5, 'dekametre': 2},
          {'dare': 7},
          {'deliver': 7},
          {'bet': 1, 'depend': 6},
          {'depict': 1, 'describe': 6},
```

Figure 9: Output of the similarity count per synonym

The output depicts the number of words from other languages that the english word is close to, based on this output we either keep the english word or we replace it with the synonym with the most similarity hits.

	A	B	C	D	E
1		Vertice1	Vertice2	Link_Strength	Score
2	0	of	fon	Strong	0.4
3	1	in	im	Strong	0.5
4	2	is	is	Strong	1
5	3	it	est	Strong	0.4
6	4	he	er	Strong	0.5
7	5	was	waar	Strong	0.5714285714
8	6	with	mit	Strong	0.5714285714
9	7	have	haben	Strong	0.6666666667
10	8	this	diese	Strong	0.4444444444
11	9	from	fon	Strong	0.5714285714
12	10	or	oder	Strong	0.6666666667
13	11	had	hatten	Strong	0.4444444444
14	12	word	Wort	Strong	0.5
15	13	we	wir	Strong	0.4
16	14	can	kannen	Strong	0.4444444444
17	15	other	andere	Strong	0.3636363636
18	16	were	wurden	Strong	0.4
19	17	all	allez	Strong	0.75
20	18	when	wann	Strong	0.5
21	19	use	benutzen	Strong	0.3636363636
22	20	an	ein	Strong	0.4
23	21	which	welche	Strong	0.5454545455
24	22	their	ihr	Strong	0.5
25	23	write	schreiben	Strong	0.4285714286
26	24	would	wurde	Strong	0.6
27	25	so	so	Strong	1

Figure 10: NodeXL computational similarity measure results

This image shows the result of edit distance similarity between every two words in our dataset, the results are then subject to a threshold that filters out only strong links between two words and stores them.

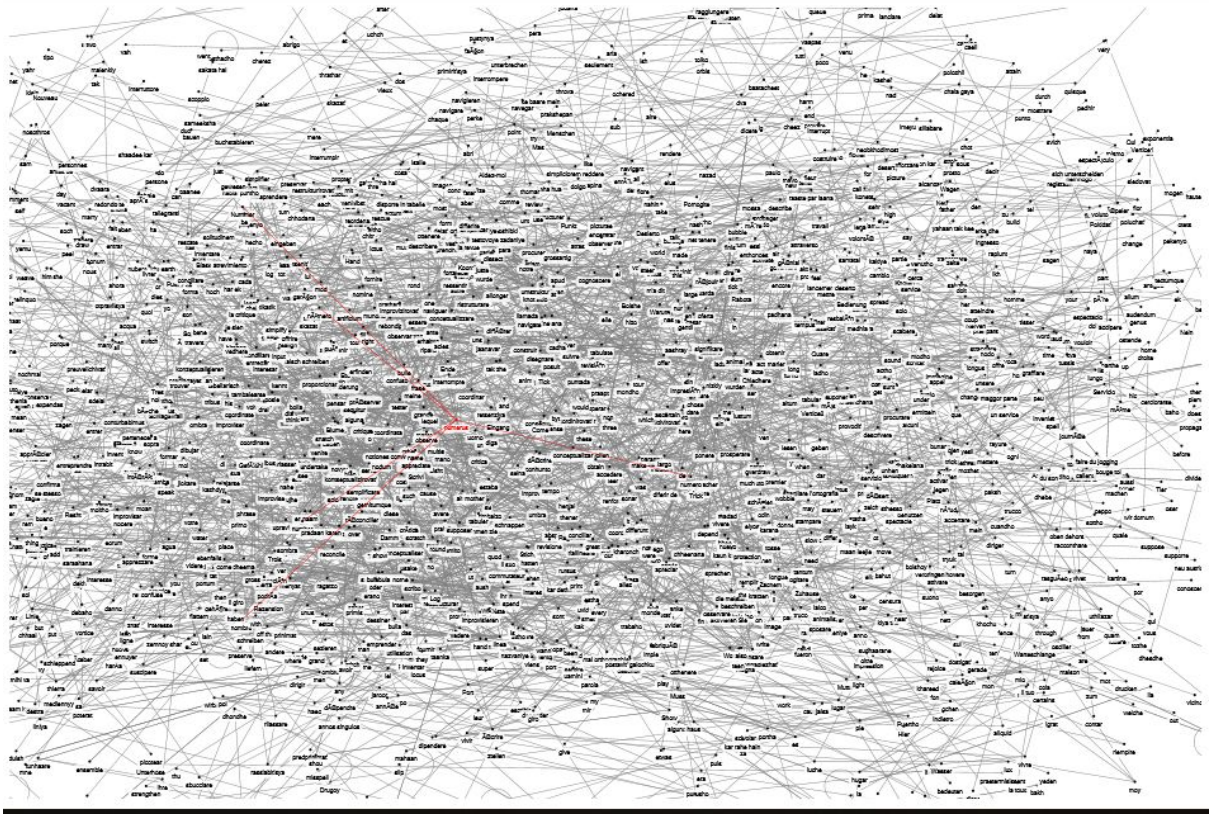

Figure 11: Graphical Representation of network on NodeXL

This figure graphically represents the word similarities on a network structure on NodeXL

Final Conclusion from NodeXL:

We arrived at the final conclusion by taking the average degree centrality of all the languages and assigning the language with the maximal average degree centrality as the most central language.

```
cd_final
```

```
{'English': 2.5066666666666667,
 'French': 2.14,
 'German': 1.6266666666666667,
 'Hindi': 1.1733333333333333,
 'Italian': 2.6566666666666667,
 'Latin': 2.3833333333333333,
 'Russian': 1.3166666666666667,
 'Spanish': 2.5366666666666666}
```

We can see that italian is the most central language from the above degree centrality computations

CHAPTER 9: CONCLUSION

This project has helped us in understanding the linguistic analysis domain, it has given us a better idea about the various challenges and methodologies that are involved in a successful attempt at linguistic analysis. It has also taught us the various approaches that can be used for any particular step during the analysis as there are no fixed or clear cut approaches defined for this field as of yet.

In our attempt at performing Linguistic analysis on the Indo-European languages and their evolution and development theory, we have come across a new approach on expanding and building up a database from scratch which will give a better quality dataset to work on. This approach, we believe, revolutionises the data collection process involved in the linguistic analysis process and can also prove to be more efficient and time saving because of its semi-automated nature.

Finally, we obtained new insights and visualizations that have helped us depict the Indo-European languages as a network like structure within which the languages have links between themselves which was not thought to have existed in the original tree like structure.

REFERENCES

[1] Renfrew, Colin. "The Origins of Indo-European Languages." Scientific American 261, no. 4 (1989): 106-15. http://www.jstor.org/stable/24987446.

[2] Boc, Alix, Anna Maria Di Sciullo, and Vladimir Makarenkov. "Classification of the Indo-European languages using a phylogenetic network approach." In Classification as a Tool for Research, pp. 647-655. Springer, Berlin, Heidelberg, 2010.

[3] Prabhu, Shreekanth. (2018). Evolving a Framework to interpret the Vedas. 10.13140/RG.2.2.32939.95529.

[4]. Clustering Semantically Equivalent Words into Cognate Sets in Multilingual Lists http://www.aclweb.org/anthology/I11-1097

[5]. Mapping the Origins and Expansion of the Indo-European Language Family Remco Bouckaert, Philippe Lemey, Science 24 Aug 2012:Vol. 337, Issue 6097, pp. 957-960 DOI: 10.1126/science.1219669

[6]. The Origins of Indo-European Languages Colin Renfrew Scientific American Vol. 261, No. 4 (OCTOBER 1989), pp. 106-115

USER MANUAL:

APPENDIX A

A.1 Definitions

The keywords given below are used in the document unambiguously. The meaning of these keywords remain as defined here, unless indicated otherwise :

1)Linguistic Analysis : Linguistics is a scientific discipline which uses different ways to study human language, cognition, mind, and their relationships.

2)Tree Model: Tree model is a model of the evolution of languages analogous to the concept of a family tree, particularly a phylogenetic tree in the biological evolution of species.

3) Proto Indo European language : Proto-Indo-European is the linguistic reconstruction of the common ancestor of the Indo-European languages, the most widely spoken language family in the world .It is by far the best understood of all proto-languages of its age.PIE is estimated to have been spoken as a single language from 4500 BC to 2500 BC.

4) Indo European languages : The Indo-European languages are a language family of several hundred related languages and dialects.The Indo-European family includes most of the modern languages of Europe; notable exceptions include Hungarian, Turkish, Finnish, Estonian, Basque, Maltese, and Sami.

5) Stopwords : Stop words usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

6)Levenshtein distance :The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words

is the minimum number of single-character edits (i.e. insertions, deletions, or substitutions) required to change one word into the other.

7) Lexical similarity : Lexical similarity is a measure of the degree to which the word sets of two given languages are similar. A lexical similarity of 1 (or 100%) would mean a total overlap between vocabularies, whereas 0 means there are no common words.

8) Cosine Similarity : Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any angle in the interval (0,π] radians.

9) Phonetics : Phonetics is a branch of linguistics that studies the sounds of human speech, or—in the case of sign languages—the equivalent aspects of sign. It is concerned with the physical properties of speech sounds or signs : their physiological production, acoustic properties, auditory perception, and neurophysiological status.

A.2 Keywords

1) Numpy : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
2) Pandas : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
3) Google translate api : Googletrans is a free and unlimited python library that implemented Google Translate API. This uses the Google Translate Ajax API to make calls to such methods as detect and translate.
4) Difflib : This module provides classes and functions for comparing sequences. It can be used for example, for comparing files, and can produce difference information in various formats, including HTML and context and unified diffs. For comparing directories and files, see also, the filecmp module.
5) Ling py: LingPy is a suite of open source Python modules for sequence comparison, distance analyses, data operations and visualization methods in quantitative historical linguistics.
6) Networkx : NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
7) Weighted Graph : A weighted graph is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive).
8) Undirected graph : An undirected graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected network. In contrast, a graph where the edges point in a direction is called a directed graph.
9) OS : An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.
10) Dataframe : A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

11) Nltk :NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

12) Nltk.wordnet : Module to find the meanings of words, synonyms, antonyms, and more. Let's cover some examples.

A.3 Common Functions

1) Os.chdir : OS module in Python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.
2) Pd.read_csv :Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analyzing data much easier.
3) Translator.translate : used to translate a word from source language to destination language as specified.
4) difflib.SequenceMatcher : This is a flexible class for comparing pairs of sequences of any type, so long as the sequence elements are hashable.
5) wordnet.synsets('rejoice') :

   WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing.

   Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept. Some of the words have only one Synset and some have several.