VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of Master of Applied Computer Science

EXPLORING ONTOLOGY DRIVEN EDITION OF DECISION TREES

Susmita Dutta

Academic year : 2019 - 2020

Promotor: Prof.Ann Nowé , Supervisor : Isel Del Carmen Grau Garcia

Faculty of Engineering

Abstract

The aim of semantic web is to elaborate simple data on the world wide web to semantic layer thereby the knowledge processing by machine becomes more simple and more shareable. Ontology is one the key technologies to understand and process semantic web. On the other hand, complex machine learning models have achieved notable success for prediction. The amalgamation of these two domains can lead to better interpretability, realistically achieve a better prediction result and understand the relationships among the attributes from ontology documents.
The main research of this thesis is to understand how machine learning algorithms can enhance the ontological features for further understanding of the important properties of ontology on a machine learning dataset. Here three main objectives were fulfilled.
1. Association from correlation among the features.
2. Class and subclass replacements based on association with correlation and without correlation.
3. Implementing ontology properties and generating an ontology file using a machine learning dataset.

Key words: Semantic Web, Ontology, RDF, Data Correlation, Pearson's Coefficient, Attributes, Ml-Schema, Decision Trees

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.
I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

I take this opportunity to acknowledge the people that have made the dissertation a success. I would like to thank my promotor, Prof. Ann Nowè for providing me with this opportunity to work on this thesis.
The next person I would like to thank is my supervisor Isel Del Carmen Grau Garcia for her unwavering guidance and support during this pandemic situation against all odds.
Finally, I am grateful to my family and friends for all the love and support throughout this process.

Preface

‘This master’s thesis came about (in part) during the period in which higher education was subjected to a lockdown and protective measures to prevent the spread of the COVID-19 virus. The process of formatting, data collection, the research method and/or other scientific work the thesis involved could therefore not always be carried out in the usual manner. The reader should bear this context in mind when reading this Master’s thesis, and also in the event that some conclusions are taken on board’.

Contents

List of Figures

List of Tables

1. Introduction

1.1. The Semantic Web

“The Semantic Web" as defined by Berners-Lee et al, “is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [1]. The vision of the future of the web is the semantic web that provides information in an explicit manner thereby making it simpler for machines to automate processes and amalgamate information in the web. The Semantic Web is known by the ability of a more meaningful representation of information for humans and computers, thereby providing a description of its contents and services in machine-readable form. It enhances services to be annotated, discovered, advertised, published and composed in an automated manner thereby facilitating interoperability and shareable features of knowledge over the web. The main purpose of Semantic Web is to make information readable by humans and computers.

The semantic web is constructed on the ability of xml to customize tagging schemes and flexible approach for data representation and with the help of ontology language the formal description of terminology used in web documents can be accomplished for the construction of semantic web.[2]

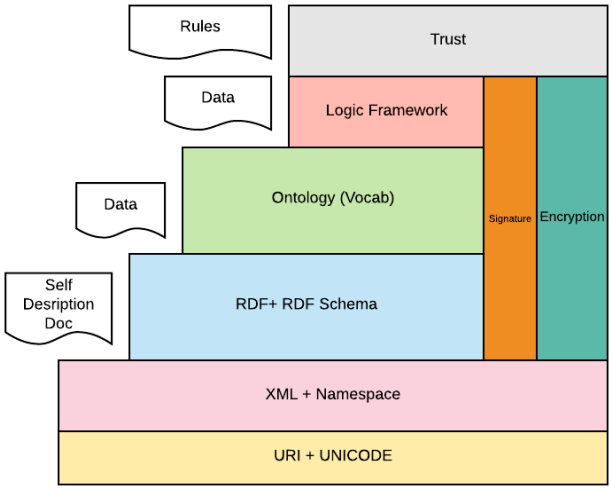

Figure 1: The architecture of the Semantic Web.

The function of semantic web is to create a machine readable document by semantically representing the data or information of the resources. The layers of the architecture of the semantic web are:

URI and Unicode: The URI also known as uniform system of identifiers to identify and locate resources on the Web. These are considered to be the building blocks of the web hence it provides a unique name to each resource. Unicode is a computing standard for the consistent encoding symbols.

Extensible Markup Language (XML): XML is a markup language with a machine-readable own format. It is a flexible text format designed to describe data and to exchange different

types of data on the Web. XML begins with a namespace declaration using the XML namespace.

Resource Descriptive Framework (RDF): RDF is the first layer of semantic web which is a framework for utilizing and representing metadata. It also describes the semantics of information about resources available on the web in a machine-readable format. RDF uses a graph model to describe the relations between the Web resources which are identified using URIs. The RDF uses RDF schema, a modelling language, to describe the classes of resources and the properties among these resources and provides a framework to infer the type of resources.

Ontology: is a language which includes a common vocabulary for published data and semantic description of the data to store the ontologies for further reference. In a simpler term it explicitly describes the semantics of the data in a uniform manner to enable communication so that both the machine and the user are able to understand.

Logic and Proof: It is important in the architecture of the semantic web, that the layers follow a logic which takes into account the structure of ontology.

Trust: The topmost layer of the semantic web to provide an assurance of the quality.

Signature and Encryption: This layer detects any form of alterations to the document and are standardized in W3C working groups.

1.2. Ontology

According to Geneserth & Nilsson, 1987 “A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.”

Later in 1993 Gruber defined the idea of ontology as an “explicit specification of conceptualization” [3].This term is taken from philosophy where Ontology is defined as a systemic account of existence. In computer science, ontology is referred to as a special kind of information object. To be more specific in the context of AI, Ontology is described as a set of representational terms of a program where the entities (classes, relation, function and other objects) with human readable text elaborate on each entity and formal axioms that constrains the interpretation and use of these terms. The main function of an ontology is to attain a common shareable knowledge that can be transferred between people and between application systems.

The W3C web ontology language (OWL) is a semantic web language designed specifically to represent ontologies. The current version of owl also referred to as “OWL 2” was developed by W3C OWL working group and published in 2009 and second edition was published in 2012.

1.2.1. Components of Ontology

There are four different components that can represent ontology: concepts, instances, relations and axioms.
Concept defines a set or class of entities or ‘things’ within a domain. Concept can be classified into two types:
Primary Concepts represents the entities which have necessary conditions in terms of their properties to belong to a class. For example, all mammals have a four chambered heart but there could be other living beings that have a four chambered heart (example crocodiles which belong to the reptile family).

Defined Concepts: represent those concepts where description is necessary and sufficient for a thing to be a member of the class. For example: all mammals are endothermic but living objects which can produce heat from within are mammals.
Relations in ontology define the interactions between concepts or between a concept's properties. Relations can be classified into two parts :
Taxonomies structure the concepts into sub- super-concept tree structures. These can also be of two kinds:
Specialisation relationships also known as the `is a kind of' relationship. For example, Ethanol is a kind of alcohol which is also a kind of an organic compound.

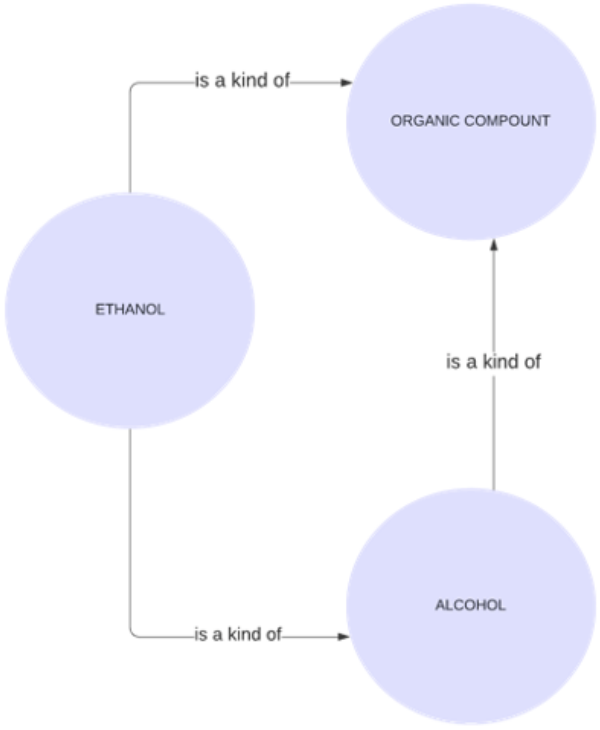

Figure 2: Specialization Relationship Diagram

Partitive relationships define concepts that are part of other concepts – like all alcohol has the $C_nH_{2n+1}OH$ structure [alcohol group].
Associative relationships functionality is to relate concepts across tree structures.
The relations can be organised into taxonomies.

Instances are the `things' shown by a concept. An ontology ideally should be void of any instances, because it is supposed to be a conceptualisation of the domain. The combination of an ontology with associated instances is known as a *knowledge base*. In certain applications ontologies can have instances (like in case of a machine learning dataset).

axioms are used to restrict values for classes or instances. the properties of relations are kinds of axioms and can be defined in ontology structure.[4]

1.2.2. Three Sub languages of OWL

Owl can be classified as three sublanguages for easy implementation by its users.

Owl lite: based on classification hierarchy and simple constraints. It is considered the lower formal complexity compared to OWL DL.

Owl dl: targets maximum expressiveness while allowing complete computational abilities and decisiveness. It includes owl language constructs which are used under certain limitations (a class cannot be an instance of another class).

Owl full: targets users who prefers maximum expression and syntactic freedom of RDF with no computational limitations.

It is highly dependent on ontology developer's choice which language will be suitable for their application to proceed forward.

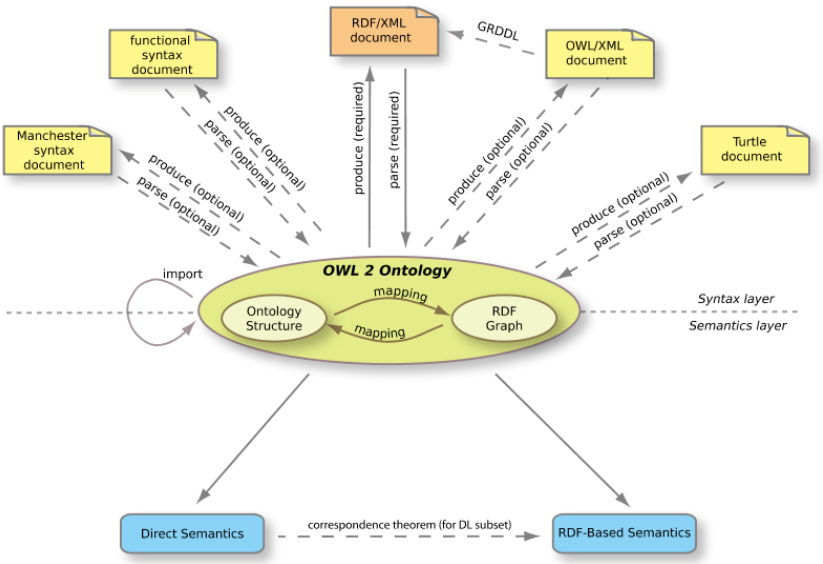

Figure 3: Different Type of OWL Syntax[5]

1.2.3. OWL Syntax

Owl2 ontologies need a constructive syntax to store information of the resources from the web. The primary exchange syntax for owl2 is RDF/XML syntax. Though this particular syntax provides a good amount of interoperability among owl2 tools other syntaxes like “Turtle”, an rdf serialization, “Owl2”, an xml serialization or Manchester syntax can also be used. Owl language is defined in a high-level structural specification which is mapped into different concrete syntaxes and there is no particular defined concrete syntax. The purpose of these syntaxes is listed below:

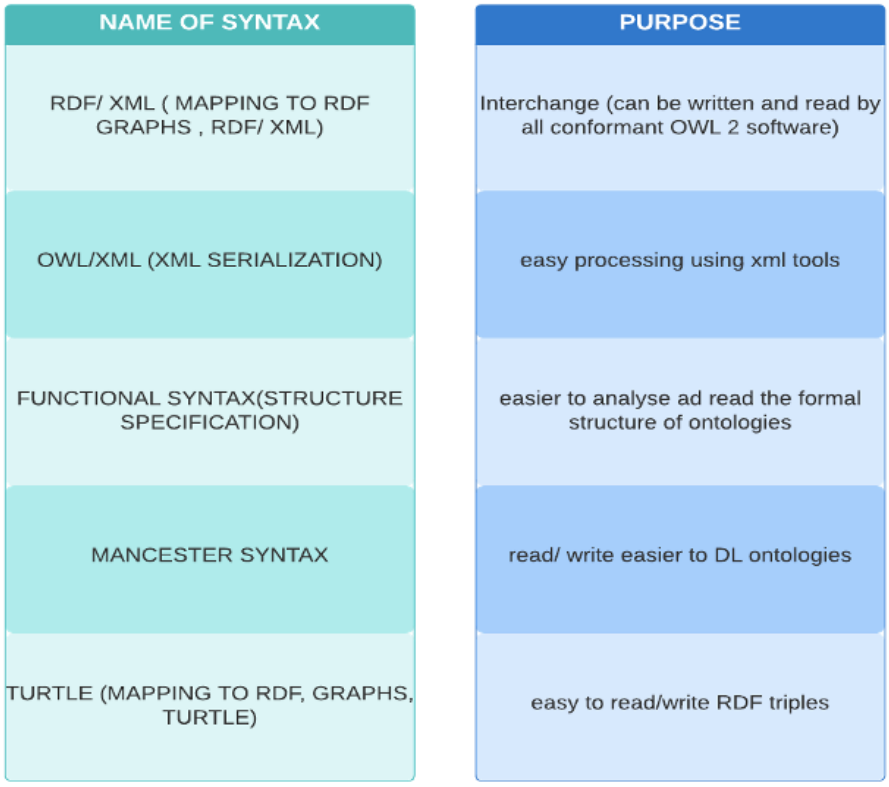

Figure 4: List of OWL Syntax [5]

DESCRIPTION OF EACH SYNTAX

THE FUNCTIONAL OWL SYNTAX:
The functional owl syntax is a text based syntax that is used as a connector between the structural specification and various concrete syntaxes. It is mainly used for translating the structural specification into other concrete syntaxes.

RDF BASED SYNTAX:
The RDF based syntax provides an xml representation of an RDF graph. This syntax is mostly supported by the Owl generating tools. A bidirectional mapping from OWL functional syntax to RDF graph is included hence the serialization into any RDF representation is less complicated.

RDF/XML:
RDF/XML is a verbose syntax which is difficult to interpret. As per literature most of the owl tools use this syntax as a default syntax to save ontologies. The advantages of RDF/XML most of the tools

published up till now are able to interpret and produce it. Along with this RDF/XML syntax can represent assertional axioms which correspond directly to triples in OWL ontologies. The verbose and the difficulty to interpret RDF/XML acts like a hindrance because the syntax is triple based. The objectification needed while translating from complex class expression, annotations and different owl axioms into triples can result in redundant texts. Parsing from one format to another always needs a substantial amount of memory because it's a bidirectional method.

TURTLE :
Turtle is considered a less complicated and considerably more readable syntax compared to RDF/XML. The advantages of Turtle are that it is more concise and human readable compared to RDF/XML. Tools and API like OWL JENA and OWL READY can produce and interpret this format.
The disadvantages of Turtle are that the syntax has the same disadvantages as the RDF / XML format.

THE MANCHESTER OWL SYNTAX :
The Manchester OWL Syntax produces a compact textual based representation for OWL ontologies and the dual functionality reading/writing is applicable .
It is mainly used for editing class expression in tools like Protégé to produce a syntax to represent a complete ontology. The advantage of Manchester Syntax that it is a compact syntax and interpretable .

OWL/XML:
OWL/XML format was designed as a complete representation format for OWL ontologies. It is derived from functional syntax thereby being more regular and simpler XML format.
The advantages of OWL/XML are that it fulfills the criteria to be an XML schema and hence it is possible to use XPath and XSLT (as per literature review). Using these tools makes it easier to read and write.
The disadvantage of OWL/XML are that the files that are generated are larger in size and hence the parsing is slower.[6]

1.2.4. Structure of Ontologies

Owl /xml can be divided into four parts:

- Namespace
- Header
- Body
- Footer

Namespace:
The namespace part is where the type of vocabulary used further is defined. The document starts with *rdf: RDF* tag because owl is an RDF document.

Header:
The Ontology Header view shows the Ontology IRI, the Ontology Version IRI and Ontology annotations for the current active ontology. The function of ontology header is to describe the resource that represents the ontology in the following parts. Though it is not mandatory that an ontology needs to include a header, but it is an introductory portion of the document to include information that will help others to understand what the rest of the document contains. This part can specify the version, compatibility information, labels and comments. In the header section a developer can specify the property *owl: imports* which refers to tools with the information required to construct a complete model of all resources referred in an ontology document. If the ontology uses elements from other ontologies a developer can declare it in this portion that the ontology is dependent on other ontologies.
Annotations are statements that explain resources using annotation properties. One of the main characteristics of annotations are that they are semantic-free properties. Any resource or axiom in an ontology can be described using these properties.[7]

Properties	Description
rdfs:label	Description of the subject resource to be implemented.
rdfs:comment	Developers comment regarding the subject resource
owl:versioninfo	Resource version
rdf:seeAlso	Information about subject resource if there is more than one.
rdfs:isDefineby	Definition the subject resource if there is more than one.

Body:

The body of the ontology contains the OWL classes, subclasses and individuals. The body also contains properties for clarity, such as: annotation, inverse, symmetric, reflexive and transitive properties.

Owl:Class
In order to describe an object, is to categorise (category) or classify (class) them having commonalities. This can be done using owl:class and rdf:type. Owl: Class represents the class containing all owl classes and the rdf:type describes the property that assigns class membership to that resource.

Owl:SubClassOf
This property restricts the membership of a class so that a taxonomic relationship can be structured from it. As mentioned above for the species class we can have different iris species as subclasses.

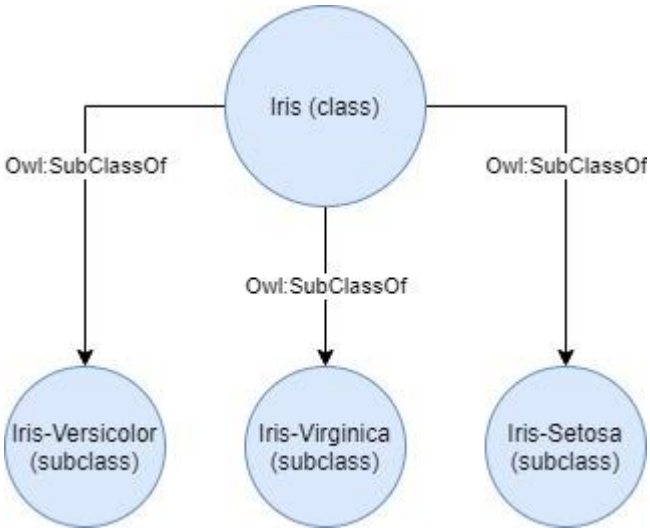

Figure 5: Owl:SubclassOf - Example

Owl:Individual
In a similar way the instances of a class are referred as owl: individuals. These are the members of a class. For example: In this thesis, a machine learning dataset is used (the iris dataset) containing five columns. The first four columns are sepal length, petal length, petal width and sepal width. The last

column contains the respective species of iris flower. Each of the first four columns contain values like sepal width has a value 5.9 unit for the species iris Sentosa. This can be broken in terms of ontology as owl:Class = sepal width, owl:Individual = 5.9.

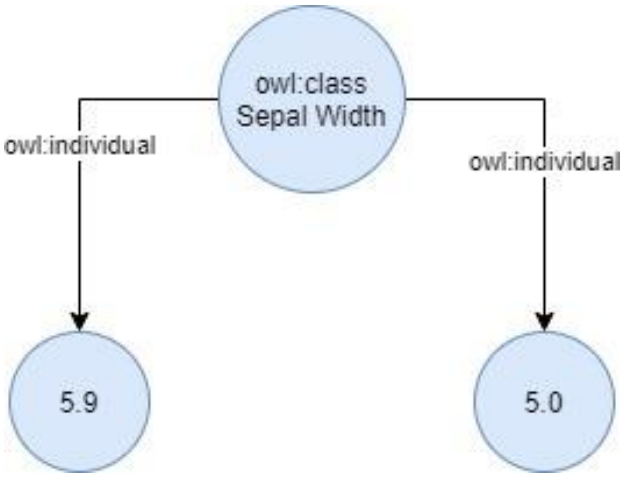

Figure 6: Owl:Individual - example

Annotation Properties:

Rdf:property:
Properties are used to define relationships between two or multiple individuals, or individuals to datavalues.
Rdf:property can be two types of property. They are object property and datatype property. These two are subclasses of rdf:property.

rdf:subpropertyof :
This annotation is not mandatory while parsing an ontology document but if needed this can be included under the rdf:property if the individual have some extra properties to define. For example, if humans not only have the property of being endothermic, but also the fact that the body temperature is always above 37 degrees approx. The temperature is considered as a subproperty in this case.

rdfs:domain
To limit a property of an individual, a domain can be specified in such scenarios.

rdfs:range
The range of a property of an individual or a class can be restricted by declaring this annotation.

Inverse Properties:

Owl:inverseOf
In owl we can show the inverse properties among relationships. In that case we need to specify it in the document.

Owl:propertyDisjointwith
Two properties of an individual / class / category can be not related to each other. In that case disjoint properties can be specified for a clarity in the owl file.

Symmetric, Reflexive and Transitive Properties:
Some additional properties can be defined in the owl file. Among these are the owl:SymmetricProperty, owl:AsymmetricProperty, owl:ReflexiveProperty, owl:IrreflexiveProperty and owl:TransitiveProperty.

In the table below the meaning of these annotations is described.[7]

owl:SymmetricProperty	(A p B) implies the statement (B p A).
owl:AsymmetricProperty	(A p B) implies there is no statement (B p A).
wl:ReflexiveProperty	Implies the statement (A p A), for all A.
owl:IrreflexiveProperty	Implies there is no statement (A p A), for all A.
owl:TransitiveProperty	(A p B) and (B p C) implies the statement (A p C)

Footer:
In the footer one needs to specify the closing of the rdf document by stating (/rdf:RDF).

1.2.5. Summary Ontology

The final goal of an ontology is to produce a readable language that can be transmitted between people and among application systems. Hence it plays an important role in interoperability on semantic web and organizations. Though there are a few cons when a huge machine learning dataset is fed to the ontology system. To negate these cons it is recommended to proceed forward and find a way where not only the domain knowledge is captured but where the creation of semantics is explicitly done in an accurate manner where the correlation among the features exist explicitly.

1.3. Data Correlation

It is considered one of the most important steps in the feature selection phase of data preprocessing more applicable when the dataset has many features and classes. It can be defined as a method to understand relationship among multiple variables in a given machine learning dataset. Hence a user can get an insight not only from one attribute depending on another attribute but one or multiple attributes are associated with other attributes in the dataset.
Correlation can be positive, negative or no correlation.

Positive correlation: occurs when feature A increases and feature B also increases or when feature A decreases and feature B also decreases. Both features have a linear relationship with each other.

Negative Correlation: happens when feature A increases and feature B decreases and vice versa.

No Correlation: No relationship between those two attributes.

Pearson's Correlation Coefficient:
In statistics, Pearson's correlation coefficient developed by Karl Pearson is one of the methods often used for a broad class of relationships among variables. It evaluates the strength of a

relationship between two vectors based on the covariance matrix of the data. Pearson's correlation coefficient between two vectors α_I and α_j is:

$$P(\alpha_i , \alpha j) = \frac{\alpha}{\sqrt{\quad \alpha}}$$

where $\text{cov}(\alpha_i , \alpha j)$ is the covariance, $\text{var}(\alpha_i)$ is the variance of α_i and $\text{var}(\alpha_j)$ is the variance of α_j .

The absolute values of Pearson's correlation coefficients are less than or equal to 1. Correlations are equal to 1 or -1 corresponding to data points lying exactly on a line, or to a bivariate distribution entirely supported on a line. The Pearson's correlation coefficient is symmetric: $P(\alpha_i , \alpha_j) = P(\alpha_j , \alpha_i)$.[8]

Feature selection is an important dimension reduction method used in many applications especially for machine learning datasets. The intention of feature selection is to search the most relevant feature subset with some criteria. Pearson's correlation has been applied in feature selection thereby producing optimum results in previous works.[9]
Pearson's correlation coefficients have also been used to measure the feature-to-feature information and feature-to-label information .[10]

1.4. Decision Trees

A decision tree classifies data objects by evaluating a series of questions about the features related with the items. Every question is contained in the node and the answers to these questions are the internal node points to every child node. Hence these series of questions form a hierarchy which is encoded in a tree. A simple yes or no question determines each internal node and a "yes" child or a "no" child. The tree is hence classified from root, which is the topmost node, following a node without children or with children ending into a leaf. Each leaf represents the appropriate target value for a class. The leaf can also have a probability vector which indicates the probability of the target attribute having a certain value. [11]

1.4.1. Algorithmic Framework for Decision Trees:

From a dataset a decision tree can be constructed using algorithms also known as decision tree inducers. The idealogy is to figure out the optimal decision tree by reducing the generalization error. Along with this other target functions can be pointed out for example reducing the number of nodes or reducing the average depth. The results generated from this indicate that using optimal decision tree algorithms is applicable only in small problems. Consequently, heuristics methods are required for solving the problem. Roughly speaking, these methods can be divided into two groups: top–down and bottom–up with clear preference in the literature to the first group. [12]

Acoording to Hancock et al 1995 figuring out a minimal decision tree with training set with consistency result can be difficult . Hence to derive a minimal equivalent decision tree for a given decision tree or building the optimal decision from decision tables in known to be np- hard.

There are different top–down decision trees algorithms likeID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), CART (Breiman *et al*., 1984). There are two conceptual phases: growing and pruning like C4.5 and CART and other algorithms perform only the growing phase. [12]

In a recursive top-down fashion decision tree learning algorithms like C4.5 (Quinlan, 1992) and CART (Breiman et al., 1984), generates a tree structure by dividing the training data into smaller subsets. All the training data are assigned at the root node and each node is split and divided into subsets . Each of the subsets are divided again and it continues till all the subsets are pure and their purity cannot be divided further. A subset is considered authentic if it contains instances of only one class. The objective is to attain few splits so that the final decision tree is small and the number of classes supporting each subject is larger. Various split selection criteria have been designed for example the information gain (Quinlan,1986b) , the gini index(Breiman et al 1984) and the gain ratio (Quinlan 1992). It measures the authenticity of a split. The learning algorithm selects the split that provides the best value for the splitting criteria at each node.[13]

Datasets with noisy data are difficult to design a purity based tree because subsets will be having faulty nodes and the isolation will never be perfect. Hence most decision tree algorithm includes a pruning technique to deal with noisy data by removing unreliable branches or subtrees. Two different techniques of pruning are commonly used:

1.4.2. Post-pruning and Pre-pruning

Post-pruning is processed after the full tree has been created and deletes those parts of the classifier that do not improve its predictive performance. Pre-pruning avoid the problem of noise by terminating the splitting process if further splitting leads to overfitting the training data. [14].

2. Historical Context

2.1. Tools for Ontology

Ontologies have been used in many domains and there has always been an increase in demand for structuring and formalizing knowledge of various fields. Previously many methods and tools were proposed and used for the design, the edition, the maintenance and the evaluation of ontologies. Two of the well-known software often used by researchers and organizations are Hermit reasoner and Protégé edition. A problem identified by Rector et al, was lack of ontology programming interface. [15]. The ontology programming interfaces are used to access and modify an ontology in a programming language. Now a days most of the programming languages are based on object -oriented programming languages hence it became an important aspect to come up with an amalgamation of ontologies and object-oriented programming language. In literatures three main strategies have been used by researchers. They are as follows:

- SPARQL
- API's (Application Programming Interfaces) like OWL API for Java
- Ontology oriented Programming

2.1.1. SPARQL

SPARQL stands for SPARQL protocol and RDF query language. Though it is a data-oriented programming language, no inference can be derived since it performs queries only. One of the major drawbacks of SPARQL is that manipulation of classes is complicated. Hence with a large machine learning dataset it will create a confusion.

2.1.2. APIs

Application Programming Interface delivers object and functions for manipulating the elements which form the ontology (classes , individuals, properties , annotation, restrictions, etc.). Some of well known APIs often used are OWLAPI and JENA in Java, owlccp for C++. Most of the APIs focus on performance of the code rather than easy usage of the library. This leads to generation of extensive source codes.

SPARQL and APIs were often used but the drawback of these were difficult to integrate with object oriented programming.

2.1.3. Ontology Oriented Programming:

The third strategy is the Ontology oriented programming. Ontology oriented programming language have the capability to reduce extensive source codes into compact form. As per literature two methods were established. Static approach where the source code in an automated fashion generates the entities available in the ontology. Dynamic approach occurs where a dynamic translation occurs between the object model and the ontology during the run time. The following table which has been partially adapted from Lamy et al [15] depicts how static object and dynamic object of ontology oriented programming and Formal ontologies match and differs.

Static object models(java)	**Dynamic object models(python)**	**Formal ontologies (OWL)**
Single inheritance	Multiple inheritance	Multiple inheritance
Disjoint classes properties are adapted automatically	Disjoint classes properties are adapted automatically	Disjoint classes need to be specified.
Class of an instance cannot be altered during run time	Class of an instance can be altered during a run time	Classes of an individual can be modified as per needed
Superclass of a class cannot be altered during run time	Superclass of a class can be altered during run time	Superclass of a class can be altered during run time as per needed
No annotation supported	No annotation supported	Annotations are supported

Table 1: Comparative study between static and dynamic objects of ontology oriented programming and Formal ontologies

2.1.4. OWLReady 2

Lamy et al came up with the python library known as OWLReady 2. The objectives of Owlready by Lamy et al was to design a module for ontology-oriented programming in Python which is compatible with OWL 2 ontologies. His main vision was to:

- providing a clear, concise and easy-to-use syntax, based on both the notations of the Protégé ontology editor and the dot notation common to many object-oriented programming languages, including Python.
- providing high-level syntactic elements for helping the manipulation of classes and a simple algorithm for local closed world reasoning. [15]

Though it is a user friendly library for generating owl file from scratch, but a few disadvantages of this library can be a bit difficult to include some extra functionalities before generating an owl file because it is not an opensource library.

2.2. Literature review: A combinatorial Execution of Ontology and Decision Tree

2.2.1. Thomopoulos et al (2010)

In 2010 Thomopoulos et al [16], came up with their work where they started from an initial domain ontology which can be obtained by semi-automated methods with the help of a domain expert. They have taken a initial learning dataset into consideration as well. The variables of the dataset match with the ontology concept. Through an iterative and interactive method learning methods and ontology based knowledge were combined. In two ways this method was evaluated, one subjective evaluation where domain experts' knowledge was taken into account to detect any form of inconsistency throughout the result. Second, objective method where numerical evaluation was taken into account. The architecture involved a separate software named COGUI . Along with this for data transformation and decision tree was implemented using R software.

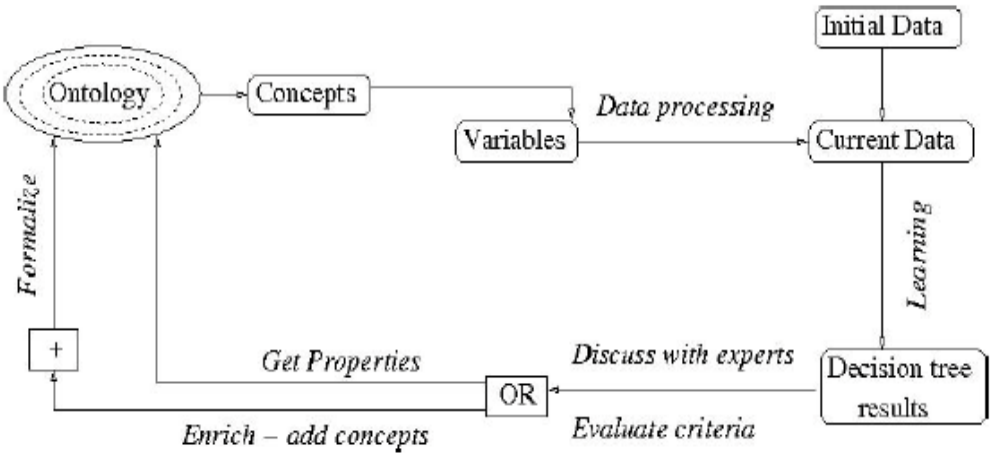

Figure 7: Model executed by Thomopoulos et al

In the paper the researchers mentioned that some of the difficult challenges they came across while developing this model was building a semi-automated method.

Some of the advantages of this model are:

- They have created a semi-automated method.
- A domain expert is involved for keeping a check on every step.
- Less of errors (because there is a constant observation).

Along with this in future work they have stated handling missing data and figuring out instances where properties are partially known would have given them a better result. Though their model might have given a better result, but the accuracy percentage isn't mentioned in their paper which makes it difficult for students to analyze its performance.

2.2.2. Grabasts et al (2015)

In 2015, Grabasts et al [16] published a paper named "Ontology-Based Classification System Development Methodology ". In this particular work the group "analysed and developed a ontology based classification system that utilizes decision tree learning with statement propositionalized attributes."

They developed a model where a raw dataset was converted into a propositionalized dataset by taking into the pseudo – attributes from the original dataset. A pseudo- attribute taxonomy was developed and then decision tree learning was implemented.
Some of the limitations as mentioned by Grabasts et al for their model were:

- The dataset should be a classification dataset
- The number of attributes and entries should be mutually proportionate.
- The dataset cannot contain null values

They implemented this work on Iris dataset and had the following results of accuracy according to the tree size: (table adapted from the paper)

Methods	Accuracy	Tree Size
Initialised dataset	96 %	11
Propositionalised dataset	89.33 %	15
J -divergence	95.33%	15

Table 2: Table adapted from paper (Grabasts et al)

Though the methodology created here used decision tree learning in ontology applications resulted in a great accuracy but some of the drawbacks can be solved. Two of them being

- Creating a methodology which is independent of dataset selection.
- A method adopted where continuous and discrete attributes will be considered and no changes will be made to the dataset.

2.2.3. Khan et al (2019)

In 2019, Khan et al [17], came with a predictive model which is a combination of Semantic Web Rule Language rules from WEKA decision tree with the help of MATLAB programming. A decision tree-based ontology model is designed in a predictive manufacturing system. For the decision tree algorithm implementation WEKA (Waikato Environment for Knowledge Analysis) was used for data pre-processing and implementing machine learning algorithm. Along with this Protégé was implemented for ontology generation.
After data processing and data cleaning, the file was exported from CSV to ARFF (attribute relation file format) to be supported by WEKA. The WEKA software implements the decision tree (J4.8 algorithm was implemented). The decision tree was utilised to create SWRL rules for the ontology model which was created in the Protégé tool. The model design by Khan et al gave an accuracy of 60.4 % thereby making it a great start for designing a manufacturing predictive model.
Some of the difficulties with this model were as follows:

- It is only prediction-based model
- Accuracy rate of the model : 60.4 %
- Based completely on software tools (Protege and weka)
- J4.8 classifier used.
- Manual entering of data during the process

Parameters	2010 Thomopoulos et al	2015 Grabats et al	2019 Khan et al
Decision Tree	C4.5	C4.5	J4.8
Ontology implementation	Before	before	After DT
Softwares	COGUI, R	MATLAB, WEKA	WEKA, PROTÉGÉ, MATLAB

Table 3: Comparison of the methodologies

2.2.4. Justicia et al (2020)

In April 2020, Justicia et al published a work on "Machine learning explainability via microaggregation and shallow decision trees ". Justicia et al created a methodology where a surrogate model of the black box model is constructed using decision tree algorithm. The surrogate model is trained on the same dataset as the black box model so that further explanation becomes easier regarding new data points classified by the same model.
The first part of the methodology computes microaggregation cluster using mean distance to average vector (MDAV) to find out the cluster representative for each section of the dataset. Once the data clustering algorithm is implemented second part of methodology deals with ontology based semantic treatment of categorical attributes. The research group at first used ontologies to capture semantics underlying categorical data in a dataset. Along with this the semantic distance between the values were calculated in order to compare and aggregate them. For the semantic distance calculation, the following equation was used:

$$distance(v_1, v_2) = \log_2\left(1 + \frac{|\phi(v_1) \cup \phi(v_2)| - |\phi(v_1) \cap \phi(v_2)|}{|\phi(v_1) \cup \phi(v_2)|}\right)$$

3. Thesis Statement

Our hypothesis is that a set of ontology properties can be established on a dataset based on numerical method without any bias among the attributes while maintaining the individual ontologies in a distributed approach .
This can be achieved by preprocessing a dataset, calculating the correlation matrix among the numeric features and generating an ontology file where the relationship among the attributes are defined.

Expected Contribution:
The proposed research is expected to contribute the following:

1. Semi-automated tool for generating ontology from machine learning dataset.
2. Curate ontology for case studies: Iris dataset, Kiva dataset (partial)
3. Explore Ontology driven edition of Decision Trees:
    i. Association from correlation
    ii. Superclass/ subclass replacements

4. Methodology

Data preprocessing

Predicting and forecasting the outcome of different datasets can be easily achieved with complex machine learning models. Though the accuracy of prediction has considerably increased and achieved success a major problem which still remains is that these models often remain uninterpretable hence affecting the safety and privacy of the users

The interpretation of datasets is one of the major research interests in the area of Semantic Web Community and knowledge engineering where the whole idea is to make the semantics of various paradigm explainable.

Publio et al, in their work have represented the idea of ML- Schema where the classes and relationship between classes are given importance. The classes in the schema represent different aspects of machine learning which includes data representation, dataset and their characteristics.

In this work a bottom – up approach was executed before ontology was generated. A well -known dataset i.e. the Iris dataset created by R. A Fisher (source 1) is used. Being a multivariate dataset with 150 instances and real attribute characteristics, preprocessing of this dataset was comparatively easier .Iris dataset is one of the recognized pattern recognition database with 5 classes and 150 instances each where each class refers to the type of iris plant.

To get to know about the data in the dataset it is important to understand the data objects, data attributes and types of data attributes. In data mining, not only knowing about the data but to find the relation between data is of utmost important. Data objects are the essential part of a dataset. A data object represents the entity. Data Objects are the group of attributes of a entity.

This is the first step of data preprocessing where the different types of attribute are classified, and the data is preprocessed. For this particular dataset , the classes are differentiated as Qualitative (nominal) and Quantitative (numeric) as attribute type where the 4 most important classes i.e. the sepal length, the sepal width , the petal length and the petal width are group as numeric attribute and the species as nominal attribute.

In the species class of the dataset, different species of Iris flower are grouped i.e. Iris-setosa, Iris-versi color, Iris-virginica.

Keeping in mind, the properties of ontology, the preprocessing of the dataset was proceeded. The minimum and the maximum cardinality i.e. the numerical value for number of instances for each column are stored in the form of list. The cardinality restrictions express the number of distinct values a property can have for a given object.

4.1. Pearson Correlation Coefficient

Before curating the ontology properties for the dataset, since a bottom-up approach is adopted, the association from correlation was taken into account.
Correlation is a method for understanding the relationship between two quantitative, continuous variables. Pearson correlation coefficient is the measure of the strength of the association between two variables. It is also known as parametric correlation test because it depends on data distribution. It can be used only when x and y are from normal distribution.
A correlation matrix on the other hand shows the correlation coefficients between set of variables. Each random variable (X_i) and in the table is correlated with each of the other values in the table (X_j). This enables us to analyze which pairs have the highest correlation.

Pearson correlation coefficient Equation:

For this specific dataset the 4 classes with numeric data are taken into consideration. Though in my code I have used panda's library to compute the correlation matrix, in general it can be executed according to this pseudo algorithm:

1 code

Algorithm 1 Pearson Correlation Coefficient

```
procedure PEARSON(a, b)
    sum = float(sum(x))
    sum = float (sum(y))
    sum_xsq =sum(map(lambda x: pow(x, 2), x))
    sum_ysq =sum(map(lambda x: pow(x, 2), y))
    psum = sum (imap(lambda x, y: x * y, x, y))
    num = psum - (sum_x * sum_y/n)
    val = pow((sum_xsq—pow(sum_x,2) / n)*(sum_ysq—pow(sum_y,2) / n), 0.5)
    if val == 0:
    return 0
    return num/ val
```

Once the Pearson coefficient correlation matrix is generated, among all the 4 classes, the diagonal and lower triangular pairs of correlation matrix are removed, and the top absolution correlation is stored in the form of a dataframe for further exploration.

4.2. Sorting Algorithm

Sorting of the correlation matrix is equally important when the dataset has large number of features. Sorting the values gives us a perspective of the strength of correlation between various features pairs in an increasing or decreasing order.
In this case, quicksort method was implemented according to this pseudo algorithm:

Algorithm 1 Quicksort Pseudocode

```
1:  procedure QUICKSORT(A, p,r)
2:    if p < r then
3:        q ← Partition(A, P, r)
4:        Quicksort(A,p,q)
5:        Quicksort(A,q+1,r)
6:        procedure PARTITION(A, p,r)
7:            x ← A[P]
8:            i ← p − 1
9:            j ← r + 1
10:
11:           while True do                repeat

13:               j ← j − 1
14:               (A[i] ← x
15:               repeat
16:                   i ← i + 1
17:               until (A[i] ¿= x)
18:               if ( then iA[j]              else
      return (j)

20:
21:
```

The given matrix is converted into a one-dimensional Series of values and then into dataframe thereby renaming the columns for better understanding.

4.3. Generating subset numeric dataframe

For a better clarity and to avoid repetition of attribute_One and attribute_Two features, Boolean indexing was executed on the dataframe. In Boolean indexing, Boolean vector to filter the data occurs. Boolean indexing is a type of indexing which uses actual values of the data in the DataFrame. In boolean indexing, data can be filtered in four ways –

- Accessing a DataFrame with a boolean index
- Applying a boolean mask to a dataframe
- Masking data based on column value
- Masking data based on index value

The dataframe is accessed using .iloc[] where a Boolean value (True or False) is passed .
Wherever the features of attribute_One and attribute_Two matches are dropped from the dataframe since their correlation coefficient is 1.0 and it doesn't give us any further explanation.
The dataframe is further pre-processed by removing duplicate rows from it and the final dataframe contains 10 rows with unique combinations along with the coefficient.

4.4. P-Value dependent/independent attribute extraction

The standard method that statisticians deploy to measure the significance of empirical analysis is known as the p- value. In order to determine the relationship among all the 5 classes from the dataset, the p – value will determine the probability that each attribute is related to each other. Also, if a p value is negative, we can conclude that the attributes are inversely related to each other.[18]

To understand how the attributes are related to each other we can enter our desired p value and check whether there are pairs of attributes above that threshold value or not. Once the pairs are obtained above the threshold value, a user can extract the attributes based on the threshold value.

Along with this when a huge machine learning dataset is considered with many features and attributes, a related attribute can be extracted based on the correlation matrix and the structure of the ontology can be redefined.

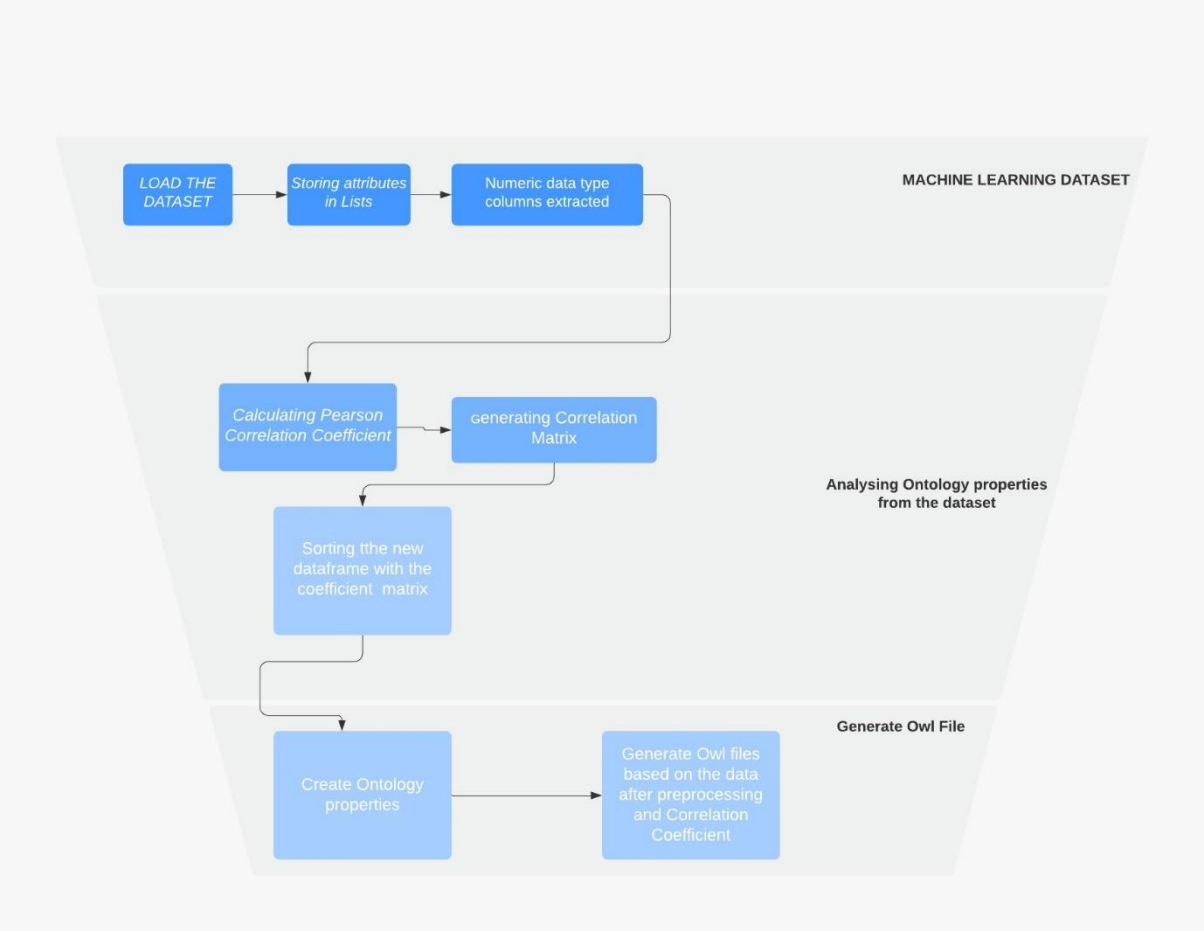

Figure 8: Full model of methodology

4.5. Parsing OWL file

After the P-value was entered In the last step of the whole methodology, the dataframe will be parsed including all the ontology properties and annotations as per the results. It is a semi -automated method and the functionalities of ontologies are based on the attributes of each dataset.

Before parsing, the functionalities of ontology are divided into three parts:

I. Class and SubClass
II. Attributes related to the superclass
III. Instance Data

Once the functionalities are executed as per the outcomes from the dataset wrangling, selective functionalities of ontology can be executed to reproduce an owl file.

5. Discussion

In the literature review it is noticed that in most of the research work either the decision tree was executed before the ontology or once the ontology is generated the decision tree was implemented on it.
In most of the cases there were restrictions regarding the type of dataset. As mentioned previously Grabasts et al restricted his model to classification datasets with number of attributes and entries mutually proportionate. Along with this no null values should be present.
In a real world machine learning dataset, these are some of the common features which are present and hence data processing is considered an utmost necessity. The data processing is done to remove null values, categorize the dataset attributes as nominal and numeric, understand and process the dataset according to domain experts' knowledge.

While designing this particular methodology, it was absolute necessary that most of the restrictions of the previous work related to ontology combined with decision tree need to be solved thereby allowing users of this model to use any kind of dataset where ontology features can be declared in an object oriented programming language and resulting in an owl syntax file which can further be used for semantic web.
As previously proposed by Publio et al of W3C Machine Learning Schema Community, ML Schema was developed as a simplistic shared schema that provides "a set of classes, properties and restrictions that can be used to represent and interchange information on Machine learning algorithms, datasets and experiments". [19] Such schemas will not only enhance the interoperability but also map to more specific ontologies and vocabularies on machine learning. Considering this factual context, the whole methodology was designed in this structure.

5.1. Results of the Iris dataset

Working on the iris dataset, where four columns specify the attributes of a particular species the conceptualization of ontology was simpler. After the data preprocessing and extracting valuable information from the dataset, we can create the taxonomy of the dataset.

Correlation matrix

The correlation matrix of all the attributes of iris dataset are shown in the below table:

Correlation Matrix

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

When a threshold value of 0.5 is considered the following three relationships are taken into account:

	attribute_One	attribute_Two	Coeff
6	sepal_length	petal_width	0.817954
8	sepal_length	petal_length	0.871754
10	petal_length	petal_width	0.962757

-

Where petal_length and petal_width have highest correlation , but in certain cases if we consider petal length and petal_width having a similarity because both the values are from the petal itself, then the second best pair of attributes can be taken into consideration which is sepal_length and petal_length.

Subclass and SuperClass replacement

This subclass or superclass replacement is possible with and without considering the Pearson correlation coefficient. A negative p–value indicates inverse relationship whereas a positive value indicates a linear relationship and this knowledge can be utilized to replace superclass/subclass in the final owl file output.

Execution of Owl functionality
Species as Class:
This functionality declares the maximum cardinality that is number of features present in the dataset and declaring Iris flower as the superclass and species as the subclass.

Attributes about Species:
In the next cell all the properties which are plausibly related to the species are declared in the form of functions.
Def attr (), defines the domain and range of the dataset
def individual() defines the different classes in the dataset (sepal_length, petal_length, sepal_width, petal_width)

Def equivalentClass() defines the property if there are any classes which are equivalent to each other (though this was not applicable for the iris dataset).

Def disjointclasses() defines if there are classes or subclasses not related at all

Some of the set operators (Complex classes) functionality can also be included while generating an OWL file.
There are intersection, union and complement. (though for this dataset they were not applicable).

Instance Data:
In this portion every row is parsed from the dataset containing the values for each class and subclass.

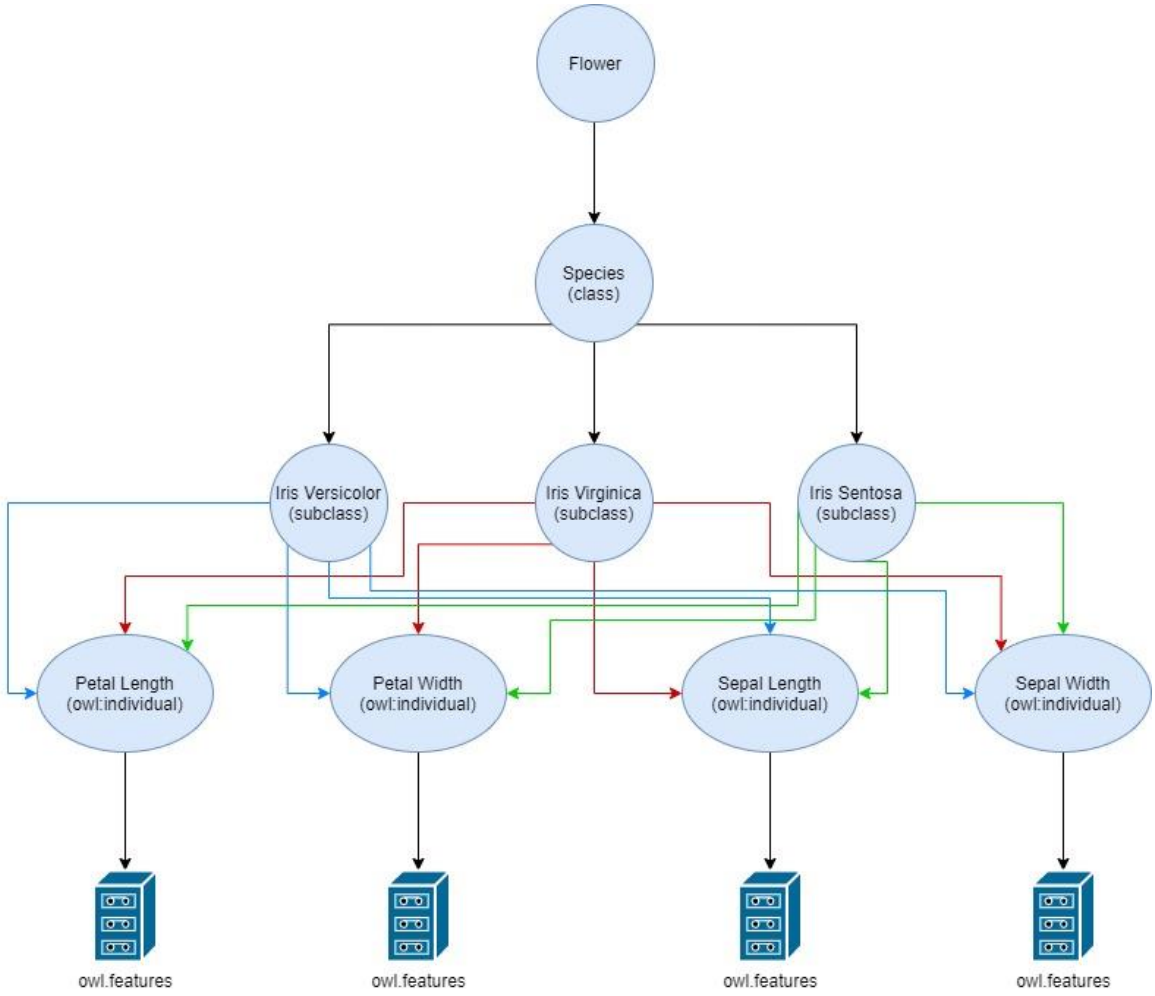

Figure 9: Graphical representation of Iris dataset

5.2. Kiva_Loans dataset [partial] result

Kiva.org is an online crowdfunding platform to support financial services to financially incapable people around the world. Kiva lenders have provided over $1 billion dollars in loans to over 2 million people. Kiva has provided a dataset of loans issued over the last two years, so that data enthusiasts can use this data as well as source external public datasets to help Kiva build models for analysing borrower welfare levels. For this particular dataset only the first CSV file is used for understanding the ontology properties.

The kiva_loans have the following columns: id, funded_amount, loan_amount, acitivity, sector, use, country_code, country, region, currency, partner_id, posted_time, disbursed_time, funded_time, term_in_months, lender_counts, tags, borrower_genders, repayment_interval, date.

Moreover there are three more related datasets in the kiva.org but this dataset have the maximum number of numeric columns hence allowing more variation for correlation matrix and implementing ontology properties.

In this dataset the country attribute is considered Super class with the property's country acode, and subproperties currency and genders.
Region is assigned to be class and Sector as subclass with activity as one of the property.
The sector contains subclass of food, transportation, arts, agriculture and entertainment.

Now these subclass can be linked to the owl:individuals which include : fund_amount, loan_amount, terms_in_months,lender_count and partner ID .

A correlation matrix can be determined from the numerical values:

```
Correlation Matrix
                funded_amount  loan_amount  partner_id  term_in_months  \
funded_amount        1.000000     0.945044   -0.075276        0.149310
loan_amount          0.945044     1.000000   -0.071251        0.184795
partner_id          -0.075276    -0.071251    1.000000        0.094878
term_in_months       0.149310     0.184795    0.094878        1.000000
lender_count         0.849168     0.798697   -0.008575        0.227283

                lender_count
funded_amount       0.849168
loan_amount         0.798697
partner_id         -0.008575
term_in_months      0.227283
lender_count        1.000000
```

The top absolute correlations among the attributes are given below

```
Top Absolute Correlations
funded_amount   loan_amount       0.945044
                lender_count      0.849168
loan_amount     lender_count      0.798697
term_in_months  lender_count      0.227283
loan_amount     term_in_months    0.184795
funded_amount   term_in_months    0.149310
partner_id      term_in_months    0.094878
funded_amount   partner_id        0.075276
loan_amount     partner_id        0.071251
partner_id      lender_count      0.008575
dtype: float64
```

This indicates larger the attributes of a dataset a correlation can be established and the owl properties can be implemented and defined in a constructive form .The superclass and subclass replacements can change the entire correlation attributes and provide us with a different result every time. This can lead to a better version of owl file with accurate identification of attributes.

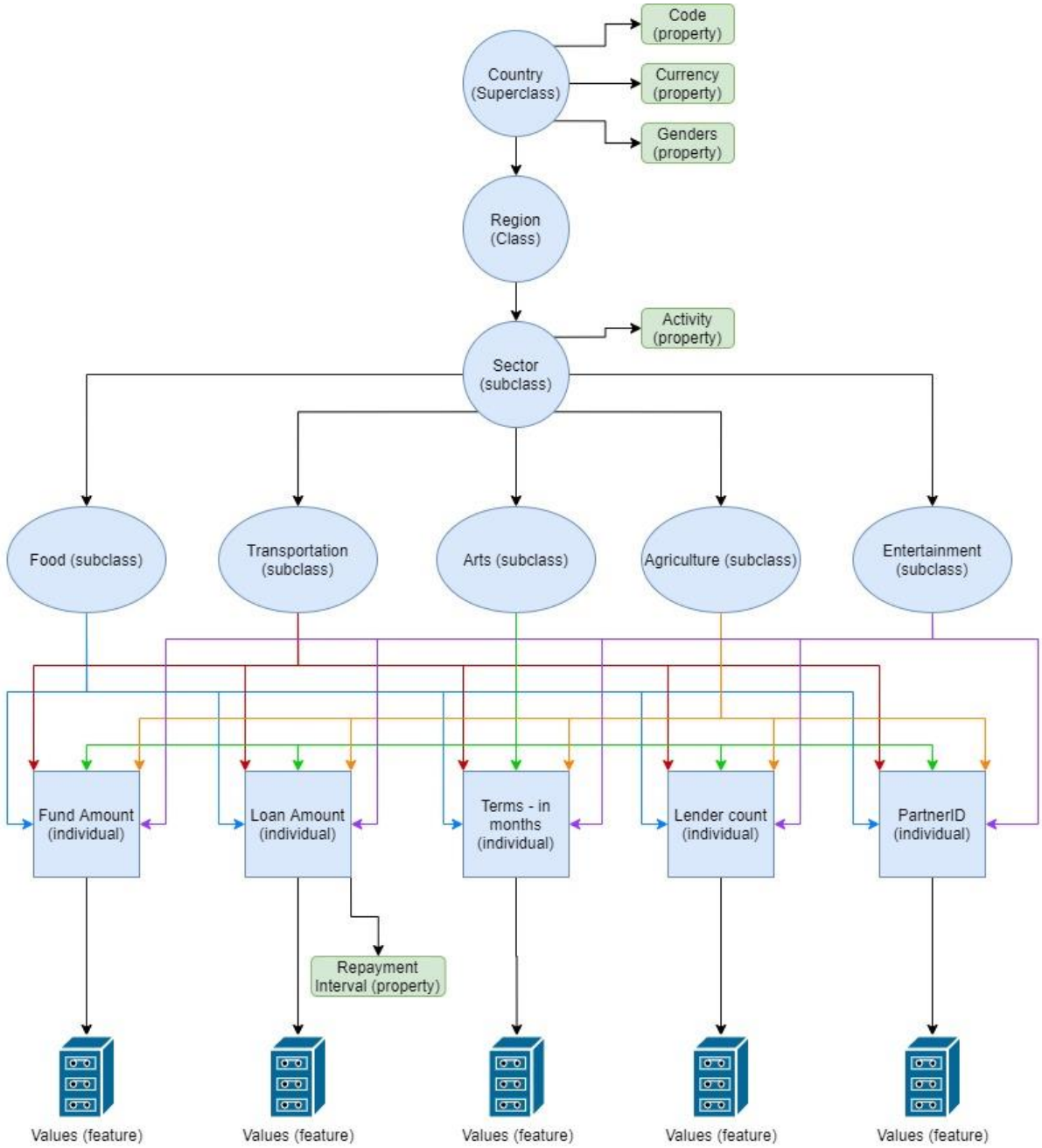

Figure 10: Graphical representation of Kiva Loans dataset

6. Conclusion

In this research a model has been presented where along with data analyzing and understanding the correlation among the attributes, an ontology file is generated where the ontology properties are implemented as well while controlling the accuracy, comprehensibility and interpretability throughout the execution. Until now most of the parts of the methodology are completely semi-automated and to some extent dataset oriented, in the future this work can be proceeded to make it more automated so that a domain expert is not necessarily needed.

The current work proceeded with Pearson's correlation coefficient before generating owl files with the ontology properties. In the future a Pearson's correlation coefficient based decision tree can be constructed which will have the capacity to handle datasets containing mixed type attributes. A splitting rule based on Pearson's correlation Coefficient can be taken into consideration to generate partitions in each node and proceed with the Decision Tree algorithm before implementing ontology properties.

7. Bibliography

[1] L. O. Berners-Lee T, Hendler J, *The Semantic Web: a New Form of Web Content That Is Meaningful to Computers Will Unleash a Revolution of New Possibilities.* Scientific American;, 2002.

[2] Mohammad Mustafa Taye, "Understanding Semantic Web and Ontologies: Theory and Applications," *Journal of Computing*, vol. 2, no. 6, 2010, [Online]. Available: https://arxiv.org/abs/1006.4567.

[3] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, doi: https://doi.org/10.1006/knac.1993.1008.

[4] "What is an Ontology?" http://www.cs.man.ac.uk/~stevensr/onto/node3.html#:~:text=The%20main%20components%20of%20an,the%20domain%20of%20molecular%20biology.

[5] "OWL SYNTAX." https://www.w3.org/TR/owl2-overview/#Ontologies.

[6] "Formal syntax description of ontologies." http://ontogenesis.knowledgeblog.org/88/.

[7] M. D. John Hebeler, Matthew Fisher, Ryan Blace, Andrew Perez-Lopez, *Semantic Web Programming*. Wiley, 2009.

[8] Y. Mu, X. Liu, and L. Wang, "A Pearson's correlation coefficient based decision tree and its parallel implementation," *Information Sciences*, vol. 435, pp. 40–58, 2018, doi: https://doi.org/10.1016/j.ins.2017.12.059.

[9] Irene Rodriguez-Lujan, "Quadratic Programming Feature Selection," *Journal of Machine Learning Research*, 2010.

[10] J. Xu, B. Tang, H. He, and H. Man, "Semisupervised Feature Selection Based on Relevance and Redundancy Criteria," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 9, pp. 1974–1984, 2017.

[11] and G. V. R. Mirambicka, A. Razia Sulthana, "Decision Tree Applied to Learning Relations between Ontologies," *Lecture Notes on Software Engineering*, vol. 2, p. 164=168, 2013.

[12] M. O. Rokach L., *Data Mining and Knowledge Discovery Handbook.* 2005.

[13] Steven L. Salzberg, *C4.5: Programs for machine learning by J. Ross Quinlan*. 1993.

[14] T. M. Isamu Shioya, "Knowledge pruning in decision trees," 2000, [Online]. Available: https://www.semanticscholar.org/paper/Knowledge-pruning-in-decision-trees-Shioya-Miura/7e2e24f21d2d894231696f7292187f842f11d3f7.

[15] J.-B. Lamy, "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies," *Artificial Intelligence in Medicine*, vol. 80, pp. 11–28, 2017, doi: https://doi.org/10.1016/j.artmed.2017.07.002.

[16] Grabusts Peter & Borisov Arkady & Aleksejeva Ludmila, "'Ontology-Based Classification System Development Methodology,'" *Information Technology and Management Science* , vol. 18, pp. 129–134, 2015.

[17] Z. M. A. Khan, S. Saeidlou, and M. Saadat, "Ontology-based decision tree model for prediction in a manufacturing network," *Production & Manufacturing Research*, vol. 7, no. 1, pp. 335–349, Jan. 2019, doi: 10.1080/21693277.2019.1621228.

[18] "P-Value Explanation." https://www.statsdirect.com/help/basics/p_values.htm.

[19] H. Correa Publio, G., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., Vanschoren, J., & Zafar, "ML-Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies," [Online]. Available: https://openreview.net/pdf?id=B1e8MrXVxQ.

Data Preprocessing :

In [23]:

```
import pandas as pd
df = pd.read_csv('iris.csv')
df.head()
df_10=pd.read_csv('iris.csv')
```

In [2]:

```
#saving all the attributes in the form of lists

dataframe_float=df.select_dtypes(include ='float64')
dataframe_float_list=list(dataframe_float.columns)
print("column list with nominal values :" '\n')
print(dataframe_float_list)

dataframe_text=df.select_dtypes(exclude ='float64')
dataframe_text_list=list(dataframe_text.columns)
print( '\n' "column list with numerical values :" '\n')
print(dataframe_text_list)
```

```
column list with nominal values :

['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

column list with numerical values :

['species']
```

In [3]:

```
column_heads=list(df)
print(column_heads)
```

```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
```

In [4]:

```
type=df["species"].unique().tolist()
print(type)
```

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

Subsetting the part of the dataframe containing numeric datatypes

In [5]:

```
dataframe_numeric= df.loc[:, 'sepal_length':'petal_width']

                #columns =['sepal_length', 'sepal_width','petal_length','petal_width'])
dataframe_numeric.head()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Max numerical value among all the features

In [6]:

```
import numpy as numpy
max_value=list(dataframe_numeric.max())
print(max_value)
```

```
[7.9, 4.4, 6.9, 2.5]
```

In [7]:

```
cardinality= list(dataframe_numeric.shape)
print(cardinality)
```

```
[150, 4]
```

Calculating the Pearson Correlation Coefficient :

In [8]:

```
print("Correlation Matrix")
correlation_mat = df.corr(method='pearson')
print(correlation_mat)
print()

def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(dataframe_numeric,10))
```

```
Correlation Matrix
              sepal_length  sepal_width  petal_length  petal_width
sepal_length      1.000000    -0.109369      0.871754     0.817954
sepal_width      -0.109369     1.000000     -0.420516    -0.356544
petal_length      0.871754    -0.420516      1.000000     0.962757
petal_width       0.817954    -0.356544      0.962757     1.000000

Top Absolute Correlations
petal_length  petal_width     0.962757
sepal_length  petal_length    0.871754
              petal_width     0.817954
sepal_width   petal_length    0.420516
              petal_width     0.356544
sepal_length  sepal_width     0.109369
dtype: float64
```

Sorting the correlation matrix If the given data has a large number of features, the correlation matrix can become very big and hence difficult to interpret. Sometimes we might want to sort the values in the matrix and see the strength of correlation between various feature pairs in an increasing or decreasing order. First, we will convert the given matrix into a one-dimensional Series of values.

In [9]:

```
corr_pairs = correlation_mat.unstack().sort_values(kind="quicksort")
print('\n', corr_pairs)
```

```
 sepal_width   petal_length   -0.420516
petal_length  sepal_width    -0.420516
sepal_width   petal_width    -0.356544
petal_width   sepal_width    -0.356544
sepal_length  sepal_width    -0.109369
sepal_width   sepal_length   -0.109369
sepal_length  petal_width     0.817954
petal_width   sepal_length    0.817954
sepal_length  petal_length    0.871754
petal_length  sepal_length    0.871754
              petal_width     0.962757
petal_width   petal_length    0.962757
sepal_length  sepal_length    1.000000
sepal_width   sepal_width     1.000000
petal_length  petal_length    1.000000
petal_width   petal_width     1.000000
dtype: float64
```

In [10]:

```
df_corr= corr_pairs.to_frame()
dist_df = df_corr.reset_index(level=[0,1])
attribute_df=dist_df.rename(columns={"level_0": "attribute_One", "level_1":
"attribute_Two",0:"Coeff"})
print(attribute_df)
```

```
   attribute_One attribute_Two     Coeff
0    sepal_width  petal_length -0.420516
1   petal_length   sepal_width -0.420516
2    sepal_width   petal_width -0.356544
3    petal_width   sepal_width -0.356544
4   sepal_length   sepal_width -0.109369
5    sepal_width  sepal_length -0.109369
6   sepal_length   petal_width  0.817954
7    petal_width  sepal_length  0.817954
8   sepal_length  petal_length  0.871754
9   petal_length  sepal_length  0.871754
10  petal_length   petal_width  0.962757
11   petal_width  petal_length  0.962757
12  sepal_length  sepal_length  1.000000
13   sepal_width   sepal_width  1.000000
14  petal_length  petal_length  1.000000
15   petal_width   petal_width  1.000000
```

In [11]:

```
print (attribute_df['attribute_One'] != attribute_df['attribute_Two']
      )
```

```
0      True
1      True
2      True
3      True
4      True
5      True
6      True
7      True
8      True
9      True
10     True
11     True
12    False
13    False
14    False
15    False
dtype: bool
```

removing the rows where attribute_One and attribute_Two matches

In [12]:

```
attribute_df = attribute_df.query("attribute_One != attribute_Two")
print(attribute_df)
```

```
   attribute_One attribute_Two     Coeff
0    sepal_width  petal_length -0.420516
1   petal_length   sepal_width -0.420516
2    sepal_width   petal_width -0.356544
3    petal_width   sepal_width -0.356544
4   sepal_length   sepal_width -0.109369
5    sepal_width  sepal_length -0.109369
6   sepal_length   petal_width  0.817954
7    petal_width  sepal_length  0.817954
8   sepal_length  petal_length  0.871754
9   petal_length  sepal_length  0.871754
10  petal_length   petal_width  0.962757
11   petal_width  petal_length  0.962757
```

removing repeated rows

In [13]:

```
result_df = attribute_df.drop_duplicates(subset=['Coeff'], keep='first')
print(result_df)
```

```
   attribute_One attribute_Two     Coeff
0    sepal_width  petal_length -0.420516
2    sepal_width   petal_width -0.356544
4   sepal_length   sepal_width -0.109369
6   sepal_length   petal_width  0.817954
8   sepal_length  petal_length  0.871754
10  petal_length   petal_width  0.962757
```

When the threshold is more than a certain coefficient value:

In [14]:

```
#p-value
threshold = 0.5
```

In [15]:

```
#if the p-value is more than 0.5 coeff
threshold_df= result_df[result_df.Coeff > threshold]
print(threshold_df)
```

```
   attribute_One attribute_Two     Coeff
6   sepal_length   petal_width  0.817954
8   sepal_length  petal_length  0.871754
10  petal_length   petal_width  0.962757
```

Result of the second attribute after the threshold value and the entered attribute is taken into account

In [17]:

```
feature_extraction_df=threshold_df.loc[result_df['attribute_One'] == column_heads[2]]
print(feature_extraction_df)
secondAttribute = list(feature_extraction_df['attribute_Two'])
print('\n','Attribute result when one attribute is entered and a related attribute is asked for '
"\n")

print(secondAttribute)
```

```
   attribute_One attribute_Two     Coeff
10  petal_length   petal_width  0.962757

 Attribute result when one attribute is entered and a related attribute is asked for

['petal_width']
```

Result of the second attribute without considering the threshold value .

In [18]:

```
feature_extraction_df=result_df.loc[result_df['attribute_One'] == column_heads[0]]
print(feature_extraction_df)
secondAttribute = list(feature_extraction_df['attribute_Two'])
print('\n','attribute result when one attribute is entered and a related attribute is asked for '
"\n")

print(secondAttribute)
```

```
  attribute_One attribute_Two     Coeff
4  sepal_length   sepal_width -0.109369
6  sepal_length   petal_width  0.817954
8  sepal_length  petal_length  0.871754

 attribute result when one attribute is entered and a related attribute is asked for

['sepal_width', 'petal_width', 'petal_length']
```

Parsing file from csv to ontology including all the preprocessed features :

In [19]:

```
#species as class
#max cardinality value as per the dataset
def speciesclass():
    return ("""<owl:Class rdf:about="species"> \n
    <rdfs:subClassOf rdf:resource="&Iris;Flower"/> \n
            <rdfs:subClassOf> \n
                <owl:Restriction> \n
                    <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">{0}</owl:maxCardinality> \n
                    <owl:onProperty rdf:resource="&features;"/> \n
                </owl:Restriction> \n
            </rdfs:subClassOf> \n
</owl:Class> """).format(cardinality[0])
print(speciesclass())
```

```
<owl:Class rdf:about="species">

    <rdfs:subClassOf rdf:resource="&Iris;Flower"/>

            <rdfs:subClassOf>

                <owl:Restriction>

                    <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">150</owl:maxCardinality>

                    <owl:onProperty rdf:resource="&features;"/>

                </owl:Restriction>

            </rdfs:subClassOf>

</owl:Class>
```

In [20]:

```
#attributes about species
#Define properties
#datatype property


def attr(column_heads):
                    return """
                    <owl:ObjectProperty rdf:ID="flower">
  <rdfs:domain rdf:resource={3}/>
```

```
  <rdfs:range rdf:resource={0}/>
</owl:ObjectProperty> """.format(column_heads[0],column_heads[1],column_heads[2],column_heads[3])

#print(attr(column_heads))

def individual(type):
    return"""
    <owl:Thing rdf:about="{0}"/>
    <owl:Thing rdf:about="{1}"/>
    <owl:Thing rdf:about="{2}"/>
    </owl:oneOf>
</owl:Class> """.format(type[0],type[1],type[2])
#print(individual(type))

#Equivalence between Classes and properties

def equivalentClass():
    return """
    <owl:Class rdf:ID="value">
  <owl:equivalentClass rdf:resource="&value1;value2"/>
</owl:Class>""".format()


#ENUMERATED CLASSES

def disjointclasses(column_heads):
                    return """
                    <owlx:DisjointClasses>
                  <owlx:Class owlx:name="#{0}" />
                  <owlx:Class owlx:name="#{1}" />
                   <owlx:Class owlx:name="#{2}" />
                    <owlx:Class owlx:name="#{3}" />
                </owlx:DisjointClasses> """.format()

#Complex Classes
#Set Operators
def intersection():
    return """<owl:Class rdf:ID="entervalue">
  <owl:intersectionOf rdf:parseType="#Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#property" />
      <owl:hasValue rdf:resource="#" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>"""


def union():
    return """
    <owl:Class rdf:ID="superclass">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#subclass" />
    <owl:Class rdf:about="#subclass" />
  </owl:unionOf>
</owl:Class>"""

def complement():
    return """

 <owl:Class rdf:ID= {0} />
  <owl:Class rdf:ID={1}>
    <owl:complementOf rdf:resource= {0} />
    <owl:Class rdf:ID={2}>
    <owl:complementOf rdf:resource={0} />
    <owl:Class rdf:ID={3}>
    <owl:complementOf rdf:resource={0} />

  </owl:Class>"""
.format(column_heads[0],secondAttribute[0],secondAttribute[1],secondAttribute[2])
#print(complement())
```

In [24]:

```
#converting my original dataframe into nested list so that each rows are parsed
```

```
listed_dataframe=df.values.tolist()
listed_dataframe_one=df_10.values.tolist()
```

In [25]:

```
#Instance Data

def property(row):
    for row in listed_dataframe_one:
        if row !=listed_dataframe_one[0]:
            i = 0
            for prop in row :
                print("""
                <species:Species rdf:about="#Features">
                    <species:feature rdf:ID = "{0}">{1} </species:feature>

            """ .format(column_heads[i],row[i]))
                i = i + 1

            print("""</owl:Class>""")
            #print(i)
#print (property(True))
```

In [26]:

```
starting_Text=("<rdf:RDF>" '\n'
"xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#" '\n'
      "xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#" '\n'
      "xmlns:owl=\"http://www.w3.org/2002/07/owl#" '\n'
      "xmlns:dc=\"http://purl.org/dc/elements/1.1/" '\n'
       "xmlns:iris=\"http://www.w3.org/iris#>" '\n'
       )

#owl header"
owlheader=("<owl:Ontology rdf:about=\"http://www.datatools.com/irisdataset>" '\n'
"<dc:title>The iris dataset Ontology</dc:title>" '\n'
"<dc:description>An ontology construction in python</dc:description>" '\n'
"</owl:Ontology>" '\n')

end_Rdf=('\n'"</rdf:RDF>")
```

In [27]:

```
def ontology():
    print(starting_Text,owlheader)
    print(speciesclass())
    print(attr(column_heads))
    print(individual(type))
    print(complement())
    for row in listed_dataframe:
                    print(property(row))
```

In [29]:

```
#print(ontology())
```

output of file that is generated

In [ ]:

```
#output of file that is generated

import sys
file = open('irisfile.owl', 'w+')
sys.stdout = file
print(ontology(),end_Rdf)
file.close()
```

In [1]:

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
```

In [2]:

```
pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 999
from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"
```

Data Processing

In [3]:

```
import pandas as pd
df = pd.read_csv('kiva_loans.csv')
df.head()
```

Out[3]:

	id	funded_amount	loan_amount	activity	sector	use	country_code	country	region	currency	partne
0	653051	300.0	300.0	Fruits & Vegetables	Food	To buy seasonal, fresh fruits to sell.	PK	Pakistan	Lahore	PKR	2
1	653053	575.0	575.0	Rickshaw	Transportation	to repair and maintain the auto rickshaw used ...	PK	Pakistan	Lahore	PKR	2
2	653068	150.0	150.0	Transportation	Transportation	To repair their old cycle-van and buy another ...	IN	India	Maynaguri	INR	3
3	653063	200.0	200.0	Embroidery	Arts	to purchase an embroidery machine and a variet...	PK	Pakistan	Lahore	PKR	2
4	653084	400.0	400.0	Milk Sales	Food	to purchase one buffalo.	PK	Pakistan	Abdul Hakeem	PKR	2

In [4]:

```
df.shape
```

Out[4]:

```
(671205, 20)
```

In [5]:

```
mpiRegion_df=pd.read_csv("kiva_mpi_region_locations.csv")
mpiRegion_df.head()
```

Out[5]:

	LocationName	ISO	country	region	world_region	MPI	geo	lat	lon

	LocationName	ISO	country	region	world_region	MPI	geo	lat	lon
0	Badakhshan Afghanistan	AFG	Afghanistan	Badakhshan	South Asia	0.367	(36.7347725, 70.81199529999999)	36.734772	70.811995
1	Badghis, Afghanistan	AFG	Afghanistan	Badghis	South Asia	0.466	(35.1671339, 63.7695384)	35.167134	63.769538
2	Baghlan, Afghanistan	AFG	Afghanistan	Baghlan	South Asia	0.300	(35.8042947, 69.2877535)	35.804295	69.287754
3	Balkh, Afghanistan	AFG	Afghanistan	Balkh	South Asia	0.301	(36.7550603, 66.8975372)	36.755060	66.897537
4	Bamyan, Afghanistan	AFG	Afghanistan	Bamyan	South Asia	0.325	(34.8100067, 67.8212104)	34.810007	67.821210

In [6]:

```
mpiRegion_df.shape
```

Out[6]:

```
(2772, 9)
```

In [7]:

```
loantheme_df=pd.read_csv("loan_theme_ids.csv")
#loantheme_df.head()

print(loantheme_df)
```

```
             id    Loan Theme ID          Loan Theme Type  Partner ID
0        638631  a1050000000skGl                  General       151.0
1        640322  a1050000000skGl                  General       151.0
2        641006  a1050000002X1ij         Higher Education       160.0
3        641019  a1050000002X1ij         Higher Education       160.0
4        641594  a1050000002VbsW  Subsistence Agriculture       336.0
...         ...              ...                      ...         ...
779087  1444237  a1050000000wf0h                  General       136.0
779088  1444238  a1050000000wf0h                  General       136.0
779089  1444240  a1050000000wf0h                  General       136.0
779090  1444241  a1050000000wf22                  General       245.0
779091  1444243  a1050000000wf22                  General       245.0

[779092 rows x 4 columns]
```

In [8]:

```
loantheme_df.shape
```

Out[8]:

```
(779092, 4)
```

In [9]:

```
loan_themes_by_region_df=pd.read_csv("loan_themes_by_region.csv")
loan_themes_by_region_df.head()
```

Out[9]:

	Partner ID	Field Partner Name	sector	Loan Theme ID	Loan Theme Type	country	forkiva	region	geocode_old	ISO	number	amount
0	9	KREDIT Microfinance Institution	General Financial Inclusion	a1050000000slfi	Higher Education	Cambodia	No	Banteay Meanchey	(13.75, 103.0)	KHM	1	450
1	9	KREDIT Microfinance Institution	General Financial Inclusion	a10500000068jPe	Vulnerable Populations	Cambodia	No	Battambang Province	NaN	KHM	58	20275
2	9	KREDIT Microfinance Institution	General Financial Inclusion	a1050000000slfi	Higher Education	Cambodia	No	Battambang Province	NaN	KHM	7	9150
3	9	KREDIT Microfinance Institution	General Financial Inclusion	a10500000068jPe	Vulnerable Populations	Cambodia	No	Kampong Cham Province	(12.0, 105.5)	KHM	1383	604950

	Partner ID	Field Partner Name	sector	Loan Theme ID	Loan Theme Type	country	forkiva	region	geocode_old	ISO	number	amount
4	9	KREDIT Microfinance Institution	General Financial Inclusion	a1050000002X1Uu	Sanitation	Cambodia	No	Kampong Cham Province	(12.0, 105.5)	KHM	3	275

In [10]:

```
loan_themes_by_region_df.shape
```

Out[10]:

```
(15736, 21)
```

In [11]:

```
listed_dataframe=df.values.tolist()

#this matches with csv in list
#print(listed_dataframe)
```

In [12]:

```
#converting the lender count column into numeric column for further analysis
df["lender_count"] = pd.to_numeric(df["lender_count"], downcast="float")
```

In [13]:

```
dataframe_float=df.select_dtypes(include ='float64')
#print(dataframe_float)
dataframe_float_list=list(dataframe_float.columns)
print(dataframe_float_list)

dataframe_text=df.select_dtypes(exclude ='float64')
#print(dataframe_text)
dataframe_text_list=list(dataframe_text.columns)
print(dataframe_text_list)
```

```
['funded_amount', 'loan_amount', 'partner_id', 'term_in_months']
['id', 'activity', 'sector', 'use', 'country_code', 'country', 'region', 'currency',
'posted_time', 'disbursed_time', 'funded_time', 'lender_count', 'tags', 'borrower_genders',
'repayment_interval', 'date']
```

In [14]:

```
#kiva_loans
column_heads=list(df)
print(column_heads)
```

```
['id', 'funded_amount', 'loan_amount', 'activity', 'sector', 'use', 'country_code', 'country', 're
gion', 'currency', 'partner_id', 'posted_time', 'disbursed_time', 'funded_time', 'term_in_months',
'lender_count', 'tags', 'borrower_genders', 'repayment_interval', 'date']
```

In [15]:

```
dataframe_float=df.select_dtypes(include ='float')
dataframe_float_list=list(dataframe_float.columns)
print("column list with nominal values :" '\n')
print(dataframe_float_list)

dataframe_text=df.select_dtypes(exclude ='float64')
dataframe_text_list=list(dataframe_text.columns)
print( '\n' "column list with numerical values :" '\n')
print(dataframe_text_list)
```

```
column list with nominal values :

['funded_amount', 'loan_amount', 'partner_id', 'term_in_months']
```

```
column list with numerical values :

['id', 'activity', 'sector', 'use', 'country_code', 'country', 'region', 'currency',
'posted_time', 'disbursed_time', 'funded_time', 'lender_count', 'tags', 'borrower_genders',
'repayment_interval', 'date']
```

In [16]:

```
mpiRegion=list(mpiRegion_df)
print(mpiRegion)
```

```
['LocationName', 'ISO', 'country', 'region', 'world_region', 'MPI', 'geo', 'lat', 'lon']
```

In [17]:

```
loantheme=list(loantheme_df)
print(loantheme)
```

```
['id', 'Loan Theme ID', 'Loan Theme Type', 'Partner ID']
```

In [18]:

```
loan_themes_by_region=list(loan_themes_by_region_df)
print(loan_themes_by_region)
```

```
['Partner ID', 'Field Partner Name', 'sector', 'Loan Theme ID', 'Loan Theme Type', 'country',
'forkiva', 'region', 'geocode_old', 'ISO', 'number', 'amount', 'LocationName', 'geocode', 'names',
'geo', 'lat', 'lon', 'mpi_region', 'mpi_geo', 'rural_pct']
```

In [19]:

```
df[df.loan_amount == 100000]
```

Out[19]:

	id	funded_amount	loan_amount	activity	sector	use	country_code	country	region	currency	partner_id	p
70499	722883	100000.0	100000.0	Agriculture	Agriculture	create more than 300 jobs for women and farmer...	HT	Haiti	Les Cayes	USD	315.0	19:

In [20]:

```
df[(df.loan_amount >= 50000) & (df.funded_amount >= 50000)]
```

Out[20]:

	id	funded_amount	loan_amount	activity	sector	use	country_code	country	region
34196	687045	50000.0	50000.0	Renewable Energy Products	Retail	to buy and sell Barefoot Power's Solar Lightin...	PE	Peru	Arequipa
43182	695450	50000.0	50000.0	Renewable Energy Products	Retail	To buy and sell Barefoot Power's solar lightin...	KE	Kenya	Nairobi
53634	706146	50000.0	50000.0	Renewable Energy Products	Retail	To buy and sell Barefoot Power solar lighting.	UG	Uganda	Kampala
70499	722883	100000.0	100000.0	Agriculture	Agriculture	create more than 300 jobs for women and farmer...	HT	Haiti	Les Cayes

	id	funded_amount	loan_amount	activity	sector	use	country_code	country	region
126839	777718	50000.0	50000.0	Agriculture	Agriculture	to buy and plant resin producing pine trees. T...	MX	Mexico	Cherán
163727	812995	50000.0	50000.0	Agriculture	Agriculture	to fund its growing loan book and further deve...	KE	Kenya	Nairobi
210975	859201	50000.0	50000.0	Agriculture	Agriculture	To work with 17 farming cooperatives to proces...	RW	Rwanda	Kigali
223120	870901	50000.0	50000.0	Higher education costs	Education	to provide loans and career services for the l...	MX	Mexico	Mexico City
408295	1055043	50000.0	50000.0	Clothing	Clothing	to set up a garment social business that will ...	AL	Albania	Cerrik
408465	1055190	50000.0	50000.0	Construction	Construction	NaN	PE	Peru	NaN
447374	1107992	50000.0	50000.0	Agriculture	Agriculture	to increase smallholder farmers' incomes by bu...	UG	Uganda	Kampala
490191	1150277	50000.0	50000.0	Health	Health	To purchase raw materials in order to produce ...	GH	Ghana	Accra
492809	1152957	50000.0	50000.0	Agriculture	Agriculture	to expand weather, farming information and fin...	GH	Ghana	Accra
494470	1154951	50000.0	50000.0	Agriculture	Agriculture	To pay smallholder coffee farmers in rural Ken...	KE	Kenya	Nairobi
496715	1156972	50000.0	50000.0	Agriculture	Agriculture	to fund the harvest of seeds of 6,000 smallhol...	MG	Madagascar	Tsihombe
509048	1169175	50000.0	50000.0	Poultry	Agriculture	to purchase chicken feed & a delivery vehicle ...	TZ	Tanzania	Dar es Salaam
523634	1183609	50000.0	50000.0	Health	Health	to mitigate CO2 & household air pollution, whi...	MW	Malawi	NaN
523659	1183916	50000.0	50000.0	Electronics Sales	Retail	to train & equip 200 rural merchants in Mozamb...	MZ	Mozambique	Maputo
526100	1186897	50000.0	50000.0	Renewable Energy Products	Retail	to distribute 200+ innovative & affordable pay...	ZM	Zambia	Lusaka
538248	1198658	50000.0	50000.0	Agriculture	Agriculture	to enable 5,000 additional small-holder farmer...	KE	Kenya	Nanyuki
541006	1201708	50000.0	50000.0	Goods Distribution	Wholesale	to bolster logistics of affordable water distr...	HT	Haiti	Petion-Ville
544548	1205071	50000.0	50000.0	Health	Health	to provide community trauma services in South ...	SS	South Sudan	Juba
						to distribute			

	id	funded_amount	loan_amount	activity	sector	use	country_code	country	region
548513	1209262	50000.0	50000.0	Renewable Energy Products	Retail	to distribute solar home systems throughout ru...	ZW	Zimbabwe	Harare
563074	1223392	50000.0	50000.0	Renewable Energy Products	Retail	to provide life-changing clean cookstoves and ...	KE	Kenya	Nairobi
565733	1226382	50000.0	50000.0	Agriculture	Agriculture	to pay 600 farming families 100% above market ...	EC	Ecuador	Quito
583307	1245201	50000.0	50000.0	Agriculture	Agriculture	to support 800+ farmers by improving their pro...	GT	Guatemala	Quetzaltenango
586970	1247422	50000.0	50000.0	Renewable Energy Products	Retail	to generate income to over 600 fishermen in Ta...	TZ	Tanzania	MUSOMA
604502	1266423	50000.0	50000.0	Agriculture	Agriculture	to add value and jobs to the local economy by ...	BJ	Benin	Parakou
614869	1277100	50000.0	50000.0	Furniture Making	Manufacturing	create jobs through environmentally-friendly m...	KE	Kenya	Nairobi
614922	1277084	50000.0	50000.0	Water Distribution	Services	to set up 13 new clean water businesses in nor...	GH	Ghana	Tamale
618264	1280213	50000.0	50000.0	Farming	Agriculture	to provide income opportunities in remote Indo...	ID	Indonesia	Simeulue
621860	1283951	50000.0	50000.0	Renewable Energy Products	Retail	to distribute 400 pay-as-you-go solar home sys...	KE	Kenya	Nairobi
631904	1294308	50000.0	50000.0	Agriculture	Agriculture	double cashew nut export output and hire about...	CI	Cote D'Ivoire	Kolia

In [21]:

```
print(df['activity'].unique())
```

```
['Fruits & Vegetables' 'Rickshaw' 'Transportation' 'Embroidery'
 'Milk Sales' 'Services' 'Dairy' 'Beauty Salon' 'Manufacturing'
 'Food Production/Sales' 'Wholesale' 'General Store' 'Clothing Sales'
 'Poultry' 'Tailoring' 'Sewing' 'Bakery' 'Restaurant' 'Food Stall'
 'Farming' 'Construction Supplies' 'Personal Products Sales'
 'Home Products Sales' 'Natural Medicines' 'Fish Selling'
 'Education provider' 'Shoe Sales' 'Machinery Rental' 'Butcher Shop'
 'Pigs' 'Personal Expenses' 'Food Market' 'Cosmetics Sales'
 'Personal Housing Expenses' 'Retail' 'Energy' 'Grocery Store'
 'Construction' 'Agriculture' 'Motorcycle Transport' 'Charcoal Sales'
 'Food' 'Pharmacy' 'Fishing' 'Timber Sales' 'Cattle' 'Electronics Repair'
 'Electronics Sales' 'Vehicle' 'Cafe' 'Blacksmith'
 'Higher education costs' 'Used Clothing' 'Fuel/Firewood' 'Upholstery'
 'Catering' 'Animal Sales' 'Cereals' 'Vehicle Repairs' 'Arts'
 'Cloth & Dressmaking Supplies' 'Mobile Phones' 'Spare Parts' 'Clothing'
 'Metal Shop' 'Barber Shop' 'Furniture Making' 'Crafts' 'Home Energy'
 'Home Appliances' 'Wedding Expenses' 'Taxi' 'Secretarial Services'
 'Livestock' 'Property' 'Recycling' 'Farm Supplies' 'Auto Repair'
 'Beverages' 'Plastics Sales' 'Electrical Goods' 'Carpentry' 'Photography'
 'Jewelry' 'Bricks' 'Pub' 'Phone Use Sales' 'Water Distribution'
 'Paper Sales' 'Computers' 'Liquor Store / Off-License' 'Utilities'
 'Knitting' 'Weaving' 'Party Supplies' 'Medical Clinic' 'Internet Cafe'
 'Consumer Goods' 'Cement' 'Electrician' 'Primary/secondary school costs'
 'Veterinary Sales' 'Land Rental' 'Laundry' 'Call Center' 'Perfumes'
 'Hotel' 'Motorcycle Repair' 'Movie Tapes & DVDs' 'Quarrying'
 'Personal Medical Expenses' 'Bookstore' 'Decorations Sales'
 'Recycled Materials' 'Office Supplies' 'Souvenir Sales'
```

```
 'Renewable Energy Products' 'Health' 'Printing' 'Phone Repair'
 'Traveling Sales' 'Flowers' 'Bicycle Repair' 'Entertainment'
 'Phone Accessories' 'Hardware' 'Used Shoes' 'Music Discs & Tapes' 'Games'
 'Balut-Making' 'Textiles' 'Child Care' 'Goods Distribution' 'Florist'
 'Cobbler' 'Dental' 'Bookbinding' 'Cheese Making' 'Bicycle Sales'
 'Well digging' 'Technology' 'Musical Performance' 'Waste Management'
 'Film' 'Tourism' 'Musical Instruments' 'Religious Articles'
 'Machine Shop' 'Cleaning Services' 'Sporting Good Sales' 'Patchwork'
 'Funerals' 'Air Conditioning' 'Communications' 'Adult Care'
 'Landscaping / Gardening' 'Aquaculture' 'Beekeeping' 'Event Planning'
 'Celebrations' 'Computer' 'Personal Care Products' 'Mobile Transactions']
```

In [22]:

```
df_corr= df[['funded_amount', 'loan_amount', 'partner_id', 'term_in_months','lender_count']]
```

In [23]:

```
print("Correlation Matrix")
correlation_mat = df_corr.corr(method='pearson')
print(correlation_mat)
print()

def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]
def get_below_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=True)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(df_corr,15))
```

```
Correlation Matrix
                funded_amount  loan_amount  partner_id  term_in_months  \
funded_amount        1.000000     0.945044   -0.075276        0.149310
loan_amount          0.945044     1.000000   -0.071251        0.184795
partner_id          -0.075276    -0.071251    1.000000        0.094878
term_in_months       0.149310     0.184795    0.094878        1.000000
lender_count         0.849168     0.798697   -0.008575        0.227283

                lender_count
funded_amount       0.849168
loan_amount         0.798697
partner_id         -0.008575
term_in_months      0.227283
lender_count        1.000000

Top Absolute Correlations
funded_amount   loan_amount       0.945044
                lender_count      0.849168
loan_amount     lender_count      0.798697
term_in_months  lender_count      0.227283
loan_amount     term_in_months    0.184795
funded_amount   term_in_months    0.149310
partner_id      term_in_months    0.094878
funded_amount   partner_id        0.075276
loan_amount     partner_id        0.071251
partner_id      lender_count      0.008575
dtype: float64
```

In [24]:

```
corr_pairs = correlation_mat.unstack().sort_values(kind="quicksort")
print('\n', corr_pairs)
```

```
 partner_id      funded_amount    -0.075276
funded_amount   partner_id       -0.075276
loan_amount     partner_id       -0.071251
partner_id      loan_amount      -0.071251
lender_count    partner_id       -0.008575
partner_id      lender_count     -0.008575
term_in_months  partner_id        0.094878
partner_id      term_in_months    0.094878
funded_amount   term_in_months    0.149310
term_in_months  funded_amount     0.149310
                loan_amount       0.184795
loan_amount     term_in_months    0.184795
term_in_months  lender_count      0.227283
lender_count    term_in_months    0.227283
                loan_amount       0.798697
loan_amount     lender_count      0.798697
lender_count    funded_amount     0.849168
funded_amount   lender_count      0.849168
                loan_amount       0.945044
loan_amount     funded_amount     0.945044
funded_amount   funded_amount     1.000000
partner_id      partner_id        1.000000
loan_amount     loan_amount       1.000000
term_in_months  term_in_months    1.000000
lender_count    lender_count      1.000000
dtype: float64
```

In [25]:

```
df_corr= corr_pairs.to_frame()
dist_df = df_corr.reset_index(level=[0,1])
attribute_df=dist_df.rename(columns={"level_0": "attribute_One", "level_1":
"attribute_Two",0:"Coeff"})
print(attribute_df)
```

```
     attribute_One   attribute_Two     Coeff
0       partner_id   funded_amount -0.075276
1    funded_amount      partner_id -0.075276
2      loan_amount      partner_id -0.071251
3       partner_id     loan_amount -0.071251
4     lender_count      partner_id -0.008575
5       partner_id    lender_count -0.008575
6   term_in_months      partner_id  0.094878
7       partner_id  term_in_months  0.094878
8    funded_amount  term_in_months  0.149310
9   term_in_months   funded_amount  0.149310
10  term_in_months     loan_amount  0.184795
11     loan_amount  term_in_months  0.184795
12  term_in_months    lender_count  0.227283
13    lender_count  term_in_months  0.227283
14    lender_count     loan_amount  0.798697
15     loan_amount    lender_count  0.798697
16    lender_count   funded_amount  0.849168
17   funded_amount    lender_count  0.849168
18   funded_amount     loan_amount  0.945044
19     loan_amount   funded_amount  0.945044
20   funded_amount   funded_amount  1.000000
21      partner_id      partner_id  1.000000
22     loan_amount     loan_amount  1.000000
23  term_in_months  term_in_months  1.000000
24    lender_count    lender_count  1.000000
```

In [26]:

```
dfcorr= corr_pairs.to_frame()

dist_df = dfcorr.reset_index(level=[0,1])
attribute_df=dist_df.rename(columns={"level_0": "attribute_One", "level_1":
```

```
"attribute_Two",0:"Coeff"})
#print(attribute_df)
attribute_df = attribute_df.query("attribute_One != attribute_Two")
print(attribute_df)
```

```
     attribute_One   attribute_Two     Coeff
0       partner_id   funded_amount -0.075276
1    funded_amount      partner_id -0.075276
2      loan_amount      partner_id -0.071251
3       partner_id     loan_amount -0.071251
4     lender_count      partner_id -0.008575
5       partner_id    lender_count -0.008575
6   term_in_months      partner_id  0.094878
7       partner_id  term_in_months  0.094878
8    funded_amount  term_in_months  0.149310
9   term_in_months   funded_amount  0.149310
10  term_in_months     loan_amount  0.184795
11     loan_amount  term_in_months  0.184795
12  term_in_months    lender_count  0.227283
13    lender_count  term_in_months  0.227283
14    lender_count     loan_amount  0.798697
15     loan_amount    lender_count  0.798697
16    lender_count   funded_amount  0.849168
17   funded_amount    lender_count  0.849168
18   funded_amount     loan_amount  0.945044
19     loan_amount   funded_amount  0.945044
```

In [27]:

```
result_df = attribute_df.drop_duplicates(subset=['Coeff'], keep='first')
print(result_df)
```

```
     attribute_One   attribute_Two     Coeff
0       partner_id   funded_amount -0.075276
2      loan_amount      partner_id -0.071251
4     lender_count      partner_id -0.008575
6   term_in_months      partner_id  0.094878
8    funded_amount  term_in_months  0.149310
10  term_in_months     loan_amount  0.184795
12  term_in_months    lender_count  0.227283
14    lender_count     loan_amount  0.798697
16    lender_count   funded_amount  0.849168
18   funded_amount     loan_amount  0.945044
```

In [28]:

```
threshold = 0.5
```

In [29]:

```
#if the threshold is more than 0.5 coeff
threshold_df= result_df[result_df.Coeff > threshold]
print(threshold_df)
```

```
    attribute_One  attribute_Two     Coeff
14   lender_count    loan_amount  0.798697
16   lender_count  funded_amount  0.849168
18  funded_amount    loan_amount  0.945044
```

In [30]:

```
feature_extraction_df=threshold_df.loc[result_df['attribute_Two'] == 'loan_amount']
print(feature_extraction_df)
secondAttribute = list(feature_extraction_df['attribute_One'])
print('\n','attribute result when one attribute is entered and a related attribute is asked for '
"\n")

print(secondAttribute)
```

```
    attribute_One attribute_Two     Coeff
14   lender_count    loan_amount  0.798697
```

```
14   lender_count   loan_amount  0.798697
18  funded_amount   loan_amount  0.945044

 attribute result when one attribute is entered and a related attribute is asked for

['lender_count', 'funded_amount']
```

In [31]:

```
#parsing all the rows of the dataset

def property(row):
    for row in listed_dataframe:
        if row !=listed_dataframe[0]:
            i = 0
            for prop in row :
                print("""
                    <owl:Class rdf:ID = "{0}":"{1}"
            """ .format(column_heads[i],row[i]))
                i = i + 1
            print("""</owl:Class>""")
            print(i)
#print (property(True))
```

In [32]:

```
#declaration of class and subclass
#basic elements
def subclassOf(row):
                    return"""
                    <owl:Class rdf:ID = class:"{0}"
                    <owl:Class rdf:ID = class:"{1}"
                    <owl:Class rdf:ID = class:"{2}"
                    <owl:Class rdf:ID = class:"{3}"
                <rdf:subClassOf rdf:resource="{4}"/>
                <owl:oneOf rdf:parseType="sector">
                <owl:Thing rdf:about="#"/>
                 </owl:oneOf>
            </owl:Class>""" .format(row[0], row[1], row[2], row[3], row[4])

def individual(type):
    return"""
    <owl:Thing rdf:about="{0}"/>
    <owl:Thing rdf:about="{1}"/>
    <owl:Thing rdf:about="{2}"/>
    </owl:oneOf>
</owl:Class> """.format(type[0],type[1],type[2])

#Equivalence between Classes and properties

def equivalentClass():
    return """"
    <owl:Class rdf:ID="value">
  <owl:equivalentClass rdf:resource="&value1;value2"/>
</owl:Class>""".format()


#ENUMERATED CLASSES

def disjointclasses(column_heads):
                    return """
                    <owlx:DisjointClasses>
                  <owlx:Class owlx:name="#{0}" />
                  <owlx:Class owlx:name="#{1}" />
                   <owlx:Class owlx:name="#{2}" />
                    <owlx:Class owlx:name="#{3}" />
                </owlx:DisjointClasses> """.format(column_heads[0],column_heads[1],column_heads[2],
column_heads[3])
def addProperty(prop):
                    return"""
              <owlx:ObjectProperty owlx:name="{}">
          <owlx:domain owlx:class="" />
          <owlx:range owlx:class="WineGrape" />
        </owlx:ObjectProperty>
        """.format(prop)
```

```
def functionalProperty(type):
    return """
    <owl:FunctionalProperty rdf:ID="type of sector">

    <rdfs:domain>

      <owl:Class>

        <owl:unionOf rdf:parseType="sector">

          <owl:Class rdf:about="#{0}"/>

          <owl:Class rdf:about="#{1}"/>

          <owl:Class rdf:about="#{2}"/>

        </owl:unionOf>

      </owl:Class>

    </rdfs:domain>

    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>

    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

  </owl:FunctionalProperty>""".format(type[0],type[1],type[2])
```

In [33]:

```
text=("<rdf:RDF" '\n'
"xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      "xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#"
      "xmlns:owl=\"http://www.w3.org/2002/07/owl#"
      "xmlns:dc=\"http://purl.org/dc/elements/1.1/"
       "xmlns:iris=\"http://www.w3.org/kiva#>"
       )

#owl header"
owlheader=("<owl:Ontology rdf:about=\"http://www.linkeddatatools.com/kivadataset>"
"<dc:title>The iris dataset Ontology</dc:title>"
"<dc:description>An ontology construction in python</dc:description>"
"</owl:Ontology>")
```

In [34]:

```
def addProperty(prop):
     return"""
    <owlx:ObjectProperty owlx:name="{}">
  <owlx:domain owlx:class="" />
  <owlx:range owlx:class="#definerange" />
</owlx:ObjectProperty>
""".format(prop)
```

In [35]:

```
def ontology():
    print(text,owlheader)
    for row in listed_dataframe:
                    print(property(row))
                    #print(indvidual(type))
                    #print(oneOf(type)),
                    #print(addProperty(prop)
                    #print(disjointclasses(column_heads))
```

In [36]:

```
#output of file
end_rdf=('\n'"</rdf:RDF>")
```

```
#print(ontology(),end_rdf)
```

In [ ]:

```
#output of file that is generated

import sys
file = open('kivafile.owl', 'w+')
sys.stdout = file

#print(ontology(),end_rdf)




file.close()
```

In [ ]:

```

```