SPARQL-R: EXTENDED SPARQL FOR STATISTICAL COMPUTATIONS

by

DIVYA SPANDANA MARNENI

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

With the advent of the Semantic Web, a huge number of Resource Description Framework (RDF) documents are being published into the Linked Open Data (LOD) Cloud. Some of these documents contain statistical data that encompasses information from various industries such as government, healthcare, schools, science, etc. Analysis of such datasets gives us valuable insights on the current socio-economic status and helps us in making decisions to improve the existing conditions. Our system offers support for executing statistical queries against RDF data. These queries are implemented as part of the SPARQL query language according to the SPARQL 1.1 specification. A library of functions is implemented, which can be added to user queries in order to incorporate statistical analysis on RDF data. Also, this library is deployed to Apache Jena Fuseki, which is a SPARQL server and accepts queries through a SPARQL endpoint. This adds to the usability of SPARQL-R library.

INDEX WORDS: Data Analysis, Statistical Analysis, RDF, Apache Jena, ARQ, SPARQL, User defined functions, R, extension libraries.

SPARQL-R: EXTENDED SPARQL FOR STATISTICAL COMPUTATIONS

by

DIVYA SPANDANA MARNENI

B.Tech., Jawaharlal Nehru Technological University, INDIA, 2012

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2018

SPARQL-R: EXTENDED SPARQL FOR STATISTICAL COMPUTATIONS

by

DIVYA SPANDANA MARNENI

Major Professor:	Krzysztof J. Kochut
Committee:	Hamid R. Arabnia
	Ismailcem Budak Arpinar

DEDICATION

Dedicated to my family and friends who have been my support throughout this journey.

ACKNOWLEDGEMENTS

I am very grateful to my major professor, Dr. Krzysztof J. Kochut. His motivation and direction in terms of thinking about research problems and arriving at solutions has made me come this far. I could not thank enough for his timely help, knowledge sharing and insights during my term under him. His expertise and guidance through my entire graduate studies helped me to accomplish things much easier. I am also grateful to him for his courses, Software Engineering and Enterprise Integration, which are very helpful in my endeavors.

I thank Dr. Hamid R. Arabnia, for his constant support in various aspects through my entire graduate life. Conversations with him made me understand and attain knowledge on various things. I also thank Dr. Ismailcem Budak Arpinar for his timely help as part of my thesis. Being a Teaching Assistant under Dr. Arpinar helped me gain experience on various academic aspects.

I express my deep gratitude to my family for encouraging me and supporting me in my journey so far. They have been my strength through thick and thin.

Finally, I thank The Department of Computer Science for providing a great environment for pursuing my Masters degree and serving as a medium to achieve life goals.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF FIGURES ix

CHAPTER

1 INTRODUCTION 1

1.1 Introduction 1

1.2 Contribution 3

2 BACKGROUND 4

2.1 Semantic Web 4

2.2 Resource Description Framework 4

2.3 SPARQL Protocol and RDF Query Language 6

2.4 SPARQL Query Processing 8

2.5 ARQ 9

2.6 R Programming Language 10

3 RELATED WORK 11

3.1 Jena Full Text Search 11

3.2 SPARQL Update 12

3.3 Extension of SPARQL Algebra 12

3.4 Custom Functions 13

3.5 Federated Queries 14

3.6 Spatial SPARQL Queries 14

3.7 RDF Data Cube Vocabulary 16

3.8 OpenCube Toolkit 17

3.9 Programmable Analytics for Linked Open Data 17

4 SPARQL-R EXTENSIONS 18

4.1 Motivation 18

4.2 Extensions 19

5 SPARQL-R IMPLEMENTATION 21

5.1 Architecture 21

5.2 Implementation 23

5.3 Custom Aggregates 23

5.4 Custom Aggregates Implementation 25

5.5 User Defined Functions 29

5.6 User Defined Functions Implementation 35

6 EVALUATION 39

6.1 Mean 40

6.2 Median 43

6.3 Standard Deviation 48

6.4 Variance 54

6.5 Skewness 55

6.6 Pearson Correlation 59

6.7 Spearman Correlation 64

6.8 Kendall Correlation 65

6.9 Z-test 67

6.10 T-test 71

7 CONCLUSION AND FUTURE WORK 76

REFERENCES 78

LIST OF FIGURES

Page

Figure 1: Architecture of SPARQL-R system 21

Figure 2: Histogram distribution of a sample data 34

Figure 3: Query result from Apache Jena Fuseki end-point for mean of Prevalence of major cardiovascular diseases in Behavioral Risk factors dataset 40

Figure 4: SPARQL query in R for mean where a vector of data values is given as input.41

Figure 5: SPARQL query executed in R for mean 41

Figure 6: Query result in Apache Jena Fuseki endpoint for retrieving a list of subjects with data value greater than mean value 42

Figure 7: Results in R for data value greater than mean value 42

Figure 8: Query result in Apache Jena Fuseki endpoint for median of alcohol usage among youth in Chronic Diseases dataset 43

Figure 9: SPARQL query executed in R for retrieving median value 44

Figure 10: Query result in Apache Jena Fuseki endpoint for data values greater than median 45

Figure 11: SPARQL query executed in R for data value greater than median 46

Figure 12: Query result in Apache Jena Fuseki for data values lesser than the median....47

Figure 13: SPARQL query executed in R for data values lesser than the median 48

Figure 14: Query result in Apache Jena Fuseki for standard deviation 49

Figure 15: SPARQL query result in R for standard deviation 49

Figure 16: Query result in Apache Jena Fuseki for subjects with alcohol usage value within one standard deviation 50
Figure 17: SPARQL query executed in R for subjects with alcohol usage value within one standard deviation 51
Figure 18: Query result in Apache Jena Fuseki for subjects with alcohol usage value within two standard deviations 53
Figure 19: SPARQL query executed in R for subjects with alcohol usage value within two standard deviations....................53
Figure 20: Query result in Apache Jena Fuseki for variance 54
Figure 21: SPARQL query executed in R for variance 54
Figure 22: Query result in Apache Jena Fuseki to show skewness towards left 55
Figure 23: Histogram plot of the data values for Alcohol usage among the youth in Chronic Diseases dataset....................56
Figure 24: SPARQL query executed in R that indicates left skewed data 56
Figure 25: Query result in Apache Jena Fuseki with positive skewness 57
Figure 26: Histogram plot of the data values for Binge Drinking Prevalence among adults which has a slightly longer right tail....................57
Figure 27: SPARQL query executed in R that indicates right skewed data 58
Figure 28: Skewness query result in Apache Jena Fuseki to indicate longer right tail 58
Figure 29: Histogram plot of the data values used for high confidence limit to show a longer right tail....................59
Figure 30: Pearson correlation query result in Apache Jena Fuseki for SAT dataset........60
Figure 31: Pearson correlation calculated in R for SAT dataset....................61

Figure 32: Pearson correlation query result in Apache Jena Fuseki for Automobile dataset which indicates positive correlation..62

Figure 33: Pearson correlation calculated in R using R’s inbuilt function for Automobile dataset indicating positive correlation 62

Figure 34: Pearson correlation query result in Apache Jena Fuseki for SAT dataset to show negative correlation 63

Figure 35: Pearson correlation calculated in R using R’s inbuilt function for Automobile dataset indicating negative correlation...64

Figure 36: Spearman correlation query result in Apache Jena Fuseki for Automobile dataset indicating insignificant negative correlation.......................................65

Figure 37: Spearman correlation calculated in R using R’s inbuilt function for Automobile dataset indicating insignificant negative correlation 65

Figure 38: Kendall correlation query result in Apache Jena Fuseki for Automobile dataset indicating negative correlation..66

Figure 39: Kendall correlation calculated in R using R’s inbuilt function for Automobile dataset indicating negative correlation...66

Figure 40: One sample Z test query result in Apache Jena Fuseki for Tobacco usage dataset 68

Figure 41: One sample Z test calculated in R for Tobacco usage dataset 69

Figure 42: Two samples Z test query result in Apache Jena Fuseki for Tobacco Usage dataset 70

Figure 43: Two samples Z test calculated in R for Tobacco Usage dataset 71

Figure 44: One sample T test query result in Apache Jena Fuseki for Chronic Diseases

Dataset..72

Figure 45: One sample T test calculated in R for Chronic Diseases dataset 73

Figure 46: Two samples T test query result in Apache Jena Fuseki for Youth Tobacco Survey dataset. 74

Figure 47: Two samples T test statistic in R for Youth Tobacco Survey dataset..............75

CHAPTER 1

INTRODUCTION

This thesis describes the design and implementation of SPARQL-R, a system for executing statistical queries from SPARQL using R. In this chapter, we discuss the significance of statistical analysis and how it is the basis for decision making for many organizations. Also, we explain how we can apply statistical analysis techniques for Resource Description Framework (RDF) data and give a brief overview of our approach in analyzing statistical data.

1.1 Introduction

With the advent of Semantic Web, increasing amount of data is being published as RDF data. Governments, various organizations and companies are offering their data through public end-points so that others can reuse it and produce valuable insights. These insights can further be used in solving many problems and in decision making to improve socio-economic conditions. Much of this data has statistical content that is devoid of proper analytical applications. As such, there is a danger of pushing data into a black hole.

In response to this potential threat, research community came up with various applications that help with the analysis of RDF datasets using various frameworks that work with SPARQL endpoints and also on Linked Open Data. These include processing, visualization, analytics and discovery of information in RDF data. While some applications concentrate on data collection and data publication, others concentrate on analysis and visualization of such data. Additionally, there are standards that describe and publish statistical data namely RDF Data Cube vocabulary [8].

A number of these applications are based on the idea of querying the data from an endpoint and further processing the result set in a way specific to the application. So, as part of these applications, SPARQL queries are used to query the RDF data through an endpoint or the Linked Open Data. Subsequently, the results set is loaded into a target environment which has the analytical processing capabilities and further analysis is done there. In addition to this process, various categories of other applications are there, and a predominant category is visualization tools.

The computation environments utilize statistical analysis capabilities, and a few of such environments include R, SAS and Matlab. Once the analysis is done, the resultant insights can be further utilized by the users.

However, the data size could be large and the process of taking the data to the computation environment is time consuming and ineffective. In addition to this, statistical analysis is not offered natively through SPARQL queries. Users always have to rely on other external frameworks to get insights into their data. There are a variety of such applications, one of which is LODVader [11].

Also, there are some domain or application specific querying capabilities implemented as part of SPARQL that help users fire application specific queries. These include Jena Full Text Search, Jena-jdbc, Spatial Searches with SPARQL, etc. Our implementation of a statistical function library for Apache Jena is based on these ideas of adding application specific querying capabilities.

1.2 Contribution

Our work on SPARQL-R is to offer statistical queries as part of SPARQL so that users can execute these queries directly through SPARQL endpoints and get the results displayed on the console. In order to implement SPARQL-R, we have utilized SPARQL's support for custom aggregates and user-defined functions. The main notion behind SPARQL-R is to leverage R's out of the box statistical functions and invoke them through SPARQL queries. This allows users to perform statistical analysis on RDF data and retrieve valuable insights.

The statistical functions are offered as a library of SPARQL functions. The library is available as a Java jar file, which should be included with the user's code in order to execute SPARQL queries enhanced with statistical analysis. Also, the library is deployed in Apache Jena Fuseki server which is a Web application that receives SPARQL queries through a REST endpoint.

The rest of this document is organized as follows: Chapter 2 provides a brief overview of the Semantic Web, RDF model, SPARQL, Query Processing Engines and the R system. Chapter 3 discusses work in the area of statistical analysis of RDF data and various application specific libraries implemented as part of SPARQL. Chapter 4 discusses our idea and solution to offer statistical queries using SPARQL. Chapter 5 explains the approach we took and the implementation details. Chapter 6 presents the application that we developed that is helpful in visualization of data using our statistical queries. Chapter 7 provides the conclusion and future work.

CHAPTER 2

BACKGROUND

2.1 Semantic Web

Semantic Web is a collection of standards that extends the current web and gives well defined meaning to the web documents so that applications can work in co-operation with each other [1]. As per Semantic Web standards, the data format used in Semantic Web technologies is the Resource Description Framework (RDF). This leads to what is called the Web of Data where web documents are interlinked, and in-turn support interaction and traversal through various documents. The documents comprise of machine readable metadata that denotes how the current document is interlinked to other documents. Stated in other words, Semantic Web is defined as a web of data that can be processed by machines [5]. Various applications are implemented using Semantic Web standards. The purpose of these applications is to process the data available on the web using smart agents or machines without much human interaction. Some of these applications include DBpedia, Friend of A Friend (FOAF), Simple Knowledge Organization System (SKOS) and many others.

2.2 Resource Description Framework (RDF)

RDF is part of World Wide Web Consortium (W3C) specifications for data modeling [6]. This in general is used for modeling of information. It is viewed as a data model that facilitates sharing data through the web. It does not have a schema and follows a graph-structure. The idea of RDF is to make statements about resources. Resources in

Semantic Web are represented using “Uniform Resource Identifier” URI notation. URI is a string of characters used to identify a resource. URIs uniquely identify a resource and can refer to people, places and any physical or logical entity in the real world. Using RDF, we make statements about the entities represented by the URIs. Each statement consists of a subject, a predicate and an object. The subject refers to the entity about which we are making the statement. The predicate represents the property or attribute of the subject and using the object we specify the value of the given property. Predicate can indicate relationship between a subject and a predicate as well. Subjects and predicates are represented using URIs whereas an object can be either an URI or a literal. The entire RDF document is comprised of these statements which are also called triples based on the subject-predicate-object format.

RDF data can be thought of as a directed labelled graph, and the direction is always from subjects to objects and predicates are edge labels. Such data is stored in tables in a three-column format. These are called triple store databases. Some of these frequently used RDF storage systems include OpenLink Virtuoso, Apache Jena TDB and AllegroGraph.

The RDF data is represented using various serialization formats. This enables storage, transmission and reconstruction of data effectively. Various serialization formats are in use today including Turtle, N-Triples, N-Quads, JSON-LD, N3 and RDF/XML. This RDF data can be stored in databases and many organizations are offering it as a service through application specific end points. This enables users to utilize the content in RDF and gain valuable insights from the data.

2.3 SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL [4] is a query language that can be used to query RDF data in the same way SQL is used to query relational data. This is specified as part of Semantic Web standards by W3C. The purpose of SPARQL is to retrieve and manage data stored as RDF. SPARQL queries are run against end-points that are offered by various data providers or by third party applications.

A SPARQL query is comprised of basic graph pattern which is again a set of triples. The triples in a SPARQL query and those in a RDF document are different in that SPARQL triples may consist of variables as well. All the triples in a given RDF document are evaluated against the specified triple pattern or graph pattern in the query and all the triples matching the query pattern are added to the result set and returned as results.

A SPARQL query is comprised of the following constructs:

1. Prefix declarations – to abbreviate the URIs.
2. Dataset definition – to specify what RDF graphs are being queried
3. Result clause – to specify what information should be returned from the query
4. Query pattern – to specify what to query in the underlying dataset
5. Query modifiers – to specify ordering, grouping qualifiers

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
   ...
}
# query modifiers
ORDER BY ...
```

SPARQL has various query forms that are illustrated with some examples below:

1. Select query: Select query is used to retrieve values from the RDF document, and the result is represented in a tabular format.

Query: Find all the people in Tim-Berners-Lee’s FOAF file that has names and email addresses [19].

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
    ?person foaf:name ?name .
    ?person foaf:mbox ?email .
}
```

2. Construct query: Construct query is used to extract information from an RDF document [20]. Additionally, it also converts the result set into a valid RDF document or graph.

Query: Creates triples based on who is whose grandfather.

```
PREFIX : <http://www.snee.com/ns/demo#>
CONSTRUCT { ?p :hasGrandfather ?g . }
WHERE {?p   :hasParent ?parent .
            ?parent :hasParent ?g .
```

```
        ?g     :gender  :male .
        }
```

3. Ask Query: This set of queries gives a Yes/No answer [21]. It tests whether a query pattern has a solution.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" ;
                 foaf:mbox  <mailto:alice@work.example> }
```

4. Describe Query: The Describe query is used to extract an RDF graph from the SPARQL endpoints [18]. It returns information about a resource.

Query: Describe resources that have mbox value as alice@org

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE
{
?x foaf:mbox <mailto:alice@org>
}
```

2.4 SPARQL Query Processing and ARQ

Query processing involves various phases before we get the final result. The given query is read and parsed by the engine. From the grammar, an abstract syntax tree is generated. But if the query is simple then an explicit syntax tree may not be generated. Then an operator graph with logical operators is generated. Optimizations are applied using the operator graph and a final query plan is generated. Code is generated for the selected query plan and execution is performed. Finally, the result is produced.

As there are many triple stores implemented, so there are many query processing engines and each of them has its own capabilities and application specific implementations added. Some of the prominent query processing engines include ARQ, Fuseki, OpenLink Virtuoso, Sesame, Stardog, Bigdata (R) etc., [3].

In our work, we are concentrating on ARQ which is supported by Apache Jena for processing SPARQL queries in Jena.

2.5 ARQ:

ARQ is developed in order to support SPARQL query processing in Apache Jena. ARQ processes SPARQL queries in a way similar to relational query processing where in the parsing engine parses the query, generates grammar, applies optimization rules, generates code and then executes it to produce the results. After the query is parsed, the SPARQL algebra for the query is generated. A high-level optimization is applied by introducing new operators that optimize the results. At a later stage low level storage specific optimization is applied.

ARQ at present supports SPARQL 1.1 standards and as part of this, ARQ implementation supports extension of SPARQL functions and capabilities. It has support for extension mechanisms that enable us to write custom expression functions and property functions. Also, ARQ gives us the facility to introduce our own algebra operators. These features of ARQ [8] enables users to implement application and domain specific queries. Below is a list of features that ARQ gives its users to extend SPARQL's querying capabilities.

1. SPARQL/Update
2. Extension of SPARQL Algebra
3. Support for custom filter functions
4. Property functions for custom processing of semantic relationships

5. Custom aggregates

6. Support for federated queries

7. Support for extension to other storage systems

8. Client-support for remote access to any SPARQL endpoint

9. Support to extend ARQ query execution

2.6 R Programming language

R is one of the statistical analysis software in popularity like SAS [22], SPSS [23], etc. When it comes to analytics, R has seen a phenomenal rise in the recent years. R is a software suite that facilitates data manipulation, calculation and graphical display of various results. It is efficient when it comes to data handling and graphical facilities. R is considered to be a statistics system because of its heavy usage when it comes to statistics. Its inbuilt data structures such as vectors, matrices, factors, lists and data-frames support analysis of data effectively.

CHAPTER 3

RELATED WORK

Using Apache Jena ARQ's extension capabilities, various researchers have implemented application specific libraries. Also, many frameworks are built to enable users to perform analytics on RDF data. This allows users to utilize information in RDF content and prompts more users to follow semantic web standards to publish their content. This enables web of data to increase, thus, allowing more data to be interlinked.

3.1 Jena Full Text Search

Text Search is implemented using ARQ's support for extension functions [13]. The extension functions utilize search capabilities provided by Apache Lucene or Elasticsearch. Consequently, the search mechanism is provided directly with SPARQL queries. The idea behind this is to index the documents for quick and easy retrieval. Each document is considered to be a collection of fields. The values in these fields are indexed using Lucene. When searches match the contents of a given field, a reference to the document containing that field is returned. In terms of RDF, each document is associated with a triple such that the predicate denotes the field in the document and the object denotes the value of the field. When a search is made, the subject is returned that satisfies the matching predicate field and object value.

Example:

```
PREFIX   ex: <http://www.example.org/resources#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX text: <http://jena.apache.org/text#>
```

```
SELECT ?s ?lbl
WHERE {
          ?s a ex:Product ;
          text:query (rdfs:label 'printer') ;
          rdfs:label ?lbl
}
```

Explanation of the query: The search examines all the triples with predicate rdfs:label and prepares a text query for value ‘printer’ on the rdfs:label property. It retrieves the complete label for each match.

3.2 SPARQL Update

Support for update operations has been added starting with SPARQL 1.1. SPARQL Update allows us to modify the graph store. Using the update queries, we can insert triples into the RDF graph, delete triples from a graph, load an RDF graph into store, clear an RDF graph from graph store, copy, move or add content from one RDF graph to another.

Example:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT { <http://example/egbook> dc:title  "This is an example title" } WHERE {}
```

Explanation of the query: This query inserts a triple with subject “<http://example/egbook>”, predicate “dc:title” and object with literal value as “This is an example title”.

3.3 Extension of SPARQL Algebra

SPARQL algebra specifies the semantics for SPARQL. By extending the SPARQL algebra, we can add new operators. These operators can be inserted into the query expression and evaluated using a custom query engine. With custom query engines, we can specify which high-level or low-level optimizations we can apply on the given query. Thus, SPARQL allows introduction of new operators and query optimization to be done as per user needs, using algebra extension and the capabilities of custom query engine.

3.4 Custom Functions

The ARQ query engine can be extended using our own functions. These include extensions of property functions, expression functions and describe handlers. Property functions enable additional filtering on triple pattern matching. Full Text Search in Apache Jena is implemented by extending the Property functions.

Example:

```
select ?x, ?y
where {
?x someProp1 SomeOb1
?y someProp2 SomeOb2
?x someLoc:propFunc ?y
}
```

Explanation of the query: The property function "propFunc" is used to apply filtering on the retrieved subjects, ?x and ?y in the first two lines of the "where" clause.

SPARQL supports extensions of expression functions, in addition to property functions. This enables users to query domain-specific data. Expression functions allow additional operations to be performed with filter, bind and select operations. We can add our own function library to Apache Jena, and examples include Leviathan Function Library, custom aggregates and filter functions. Leviathan function library supports queries with mathematical operators such as sin, cos, cosec, cot, tan, log, ln, pow etc.

Example:

```
SELECT (<http://example/countLiterals>(?o) AS ?x) {?s ?p ?o}
```

Explanation of the query: This query uses the user defined function that counts the number of objects that are literals.

3.5 Federated Queries

We may encounter scenarios where querying should be done across multiple data sources. These data sources are distributed over multiple endpoints. SPARQL 1.1 supports queries that merge data which may be distributed, and we query against multiple datasets using Federated queries.

Example

The shown below query uses a second RDF graph "http://example.org/myfoaf.rdf" in addition to the default document [24].

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.org/myfoaf.rdf>
WHERE
{
  <http://example.org/myfoaf/I> foaf:knows ?person .
  SERVICE <http://people.example.org/sparql> {
          ?person foaf:name ?name . }
}
```

Explanation of the query: This federated query uses both foaf rdf document and the rdf document that comes from the end-point http://people.example.org/sparql to answer the request to retrieve the names of all persons known by the resource "http://example.org/myfoaf/I".

3.6 Spatial SPARQL queries

This is yet another extension of Apache Jena ARQ that allows users to perform simple spatial searches [11]. In such scenarios, we need to work with a spatial dataset, and RDF data is supported with Geo data for indexing and querying. Two such RDF representations of Geo data are available: Latitude/Longitude format and “Well Known Text Literal” format. This geo data is loaded in spatial data set and queries are executed. Some sample queries include finding nearby places within a circle, within a box, north locations, south locations, west and east locations etc.

Example:

```
PREFIX spatial: <http://jena.apache.org/spatial#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?placeName
{
        ?place spatial:nearby (51.46 2.6 10 'km') .
        ?place rdfs:label ?placeName
}
```

Explanation of the query: This query is to find the nearby places, within a 10km radius of the given center, for all the subjects.

Given various application specific functions till now, SPARQL does not stand out in terms of post processing of resultant data. In various applications, the results from the SPARQL queries are extracted and further processed using various programming environments. This is done by converting the result set from SPARQL to native data structures in Java, Python, R, etc for any additional post-processing. An example application that follows this process is RDFReactor [7]. In this application, RDF data is packed into Java objects, and the data in the objects is queried using Java methods.

There are also applications that help with analytics by implementing a separate framework where in RDF data is fetched from Linked Open Data cloud as a stream, and is given for analytics and visualization of data through an interface. LODVader (LOD Visualization, Analytics and DiscovERY in Real-time) is one such application [11]. Using this application, one can search, download and create indexes for data in LOD cloud. Further the users can analyze the data in terms of page ranking, find similarity measures and get top-N links between two datasets etc.

3.7 RDF Data Cube Vocabulary

RDF Data cube vocabulary is a Semantic Web standard that allows users to publish multi-dimensional data and also to link it to other related data sets [8]. It has become a core foundation that supports publication of statistical data and other multi-dimensional data sets. The cube model used here can be used for other data sets and OLAP cubes as well. The cube model is comprised of dimensions, attributes and measures, which are collectively called components. The dimension components identify the observations and these dimensions are collections of related attributes. For example, a product dimension has attributes product name, product id, product category, product quality, etc. The measure represents the aggregation of numeric values that are being observed. Attributes specify the variable for which observation is being made and allow us to interpret the observed value.

Below is a sample definition of a datacube which consists of dimensions, measures and attributes.

```
eg:dsd-le a qb:DataStructureDefinition;
        # The dimensions
        qb:component [ qb:dimension eg:refArea;            qb:order 1 ];
        qb:component [ qb:dimension eg:refPeriod;          qb:order 2 ];
        qb:component [ qb:dimension sdmx-dimension:sex; qb:order 3 ];
        # The measure(s)
        qb:component [ qb:measure eg:lifeExpectancy];
        # The attributes
        qb:component [ qb:attribute sdmx-attribute:unitMeasure;
                qb:componentRequired "true"^^xsd:boolean;
                qb:componentAttachment qb:DataSet; ] .
```

3.8 OpenCube Toolkit

OpenCube Toolkit project offers a Software Development Kit that allows users to use the linked data cubes and build their own custom applications [15]. It has support to build projects under various use-cases. Also, this project supports effective utilization of linked data cubes. It solves the users' problems to expose statistical data as linked open data in a standard way. Also, users can apply advanced visualization techniques and perform complex statistical calculations.

The toolkit offers a browser that enables users to specify the attributes for construction of data cubes and further allows roll-up and drill-down operations. These cubes define an aggregation set. One of the interesting features of this toolkit is that it facilitates the usage of R within the framework. R is run as a web service using "Rserve" package in R. An R script is executed as part of this and generates a data frame which in turn is converted to RDF.

3.9 Programmable Analytics for Linked Open Data

This is an initiative towards a programmable web of Linked Open Data. Huge amount of RDF data is available as part of Linked Open Data (LOD) cloud. This project is about integrating computations on linked data with the statistical programming platform R [9]. R has a package named "SPARQL" which is helpful in querying RDF data through SPARQL endpoints. In this work, RDF resources are mapped to R variables. An RDF instance data is converted to R dataframe. Further more, the native functions in R are applied on RDF content loaded in R environment.

CHAPTER 4

SPARQL-R EXTENSIBILITY MECHANISMS

4.1 Motivation

In the existing implementations of the SPARQL query processors and various applications implemented on top of RDF modeling, there is not much support in terms of functions that natively facilitate analytical queries. These analytical queries help us to process the RDF documents and retrieve some useful insights based on the results from SPARQL queries.

At present, there are various applications to analyze numerical data. SAS, R, and Matlab are the most prominent among them. For this, users fire some preliminary SPARQL queries, get the result set from SPARQL, then load the result set into the computation environment. Users further run the queries using the programming language of their choice, using for example R, Python, Matlab, etc.

Over the past few years, there has been a significant increase in using R for data analysis. Also, R has seemingly endless support from the developer community in terms of adding new packages that are domain specific. R has various packages that help with data analysis. Some of them include mgcv, randomForest, multcomp, caret etc. There are few packages namely ‘RRDF’ and ‘SPARQL’ that help with loading of RDF data in R. Users work with it by querying locally loaded data or data available through remote end points. For this purpose, R’s native data structures (like matrix or data-frame) can be used [10]. The RDF file is loaded into R data structures and this data is queried using SPARQL.

In addition to SPARQL queries, we can also use R packages to do analytics on RDF data and retrieve useful insights. Users can follow a similar approach in environments like SAS and Matlab wherein users load the data into native data structures and analyze it using locally available functions.

4.2 Extensions

In the current work, we analyzed the existing aggregates, expression functions, and property functions and worked on extending the ARQ with statistical capabilities. This system works in reverse when compared to other analytical platforms wherein we provide native support for RDF data analysis using SPARQL queries. This has been implemented by using the SPARQL 1.1 support for extensions to the query processing engine in ARQ. This system is similar to other extension libraries like jena-jdbc, jena-spatial, and jena-text.

A function library is developed to facilitate statistical queries on numerical data. These statistical queries include mean, median, standard deviation, variance, correlations like Pearson, Spearman, Kendall and test statistics such as Z-Test, T-Test and P-Test.

These statistical functions are offered as aggregate functions and expression functions using user-defined functions [14] in SPARQL. These user-defined functions in turn use the R library functions and accomplish the given task by retrieving the results from R. The ARQ engine is written in Java. So, we are defining our functions in Java. Also, R is interoperable with Java, and this is accomplished with third party integration libraries like Java-R-Interface (JRI). Our user-defined functions in ARQ call the R methods, which invoke the statistical functions in R. These functions take as input the numerical values in the given RDF dataset and work on literals and in turn provide us the results as point values.

In addition to providing the results on the console, this module is integrated with Apache Jena Fuseki which is a rest endpoint for SPARQL queries. Due to this offering, the results of statistical queries can also be used as part of web applications.

CHAPTER 5

SPARQL-R IMPLEMENTATION

The SPARQL-R system has been implemented in the Java programming language. We used Apache Jena and Java-R Integration (JRI) libraries for adding extensions to the SPARQL queries.

5.1 Architecture

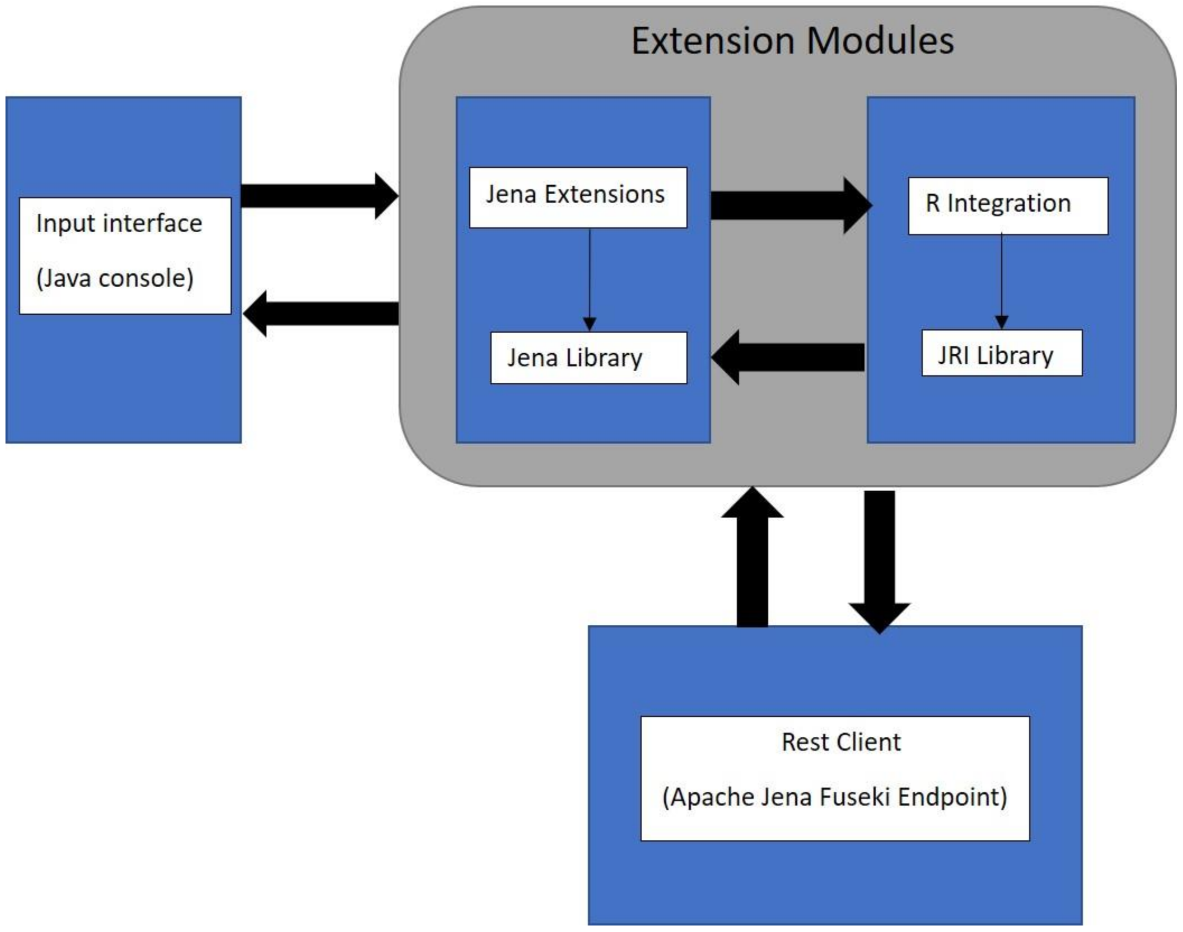


Figure 1: Architecture of SPARQL-R system.

The system is split into two modules. The first module has the implementation for SPARQL extension functions for custom aggregates and user defined functions. This module is named ‘Jena Extension’. The second module has the classes that executes R functions. This module is named ‘R-Integration.’

The R-Integration module utilizes JRI (Java-R Interface) library [12] to call and execute R functions in Java. The Rengine in JRI library is responsible for allowing us to write and execute R functions as Java statements. The reason behind using JRI is that we are leveraging the statistical functions available in R for our SPARQL statistical functions implemented in Java. This module has various classes and methods written to execute the statistical functions namely: mean, median, standard deviation, variance, skewness, z-test, t-test, pearson correlation, kendall correlation, and spearman correlation.

The ‘Jena Extension’ module allows us to execute statistical functions in SPARQL queries. Jena Extension does the below tasks:

1. Read the variables from the user query
2. Get a list of bindings for query variables
3. Aggregate the data values bound to the variables
4. Call the R statistical function
5. Retrieve the result and prepare it as a node value or RDF result set
6. Display the results in desired format for the user

In addition to writing extension modules, we have enabled the execution of queries from console application and Apache Jena Fuseki. Apache Jena Fuseki is a SPARQL server that can run as a Java web application, an operating system service or as a standalone server. Fuseki has integration with Transactional Database (TDB) to provide a persistent storage layer. We can upload the datasets to Fuseki and make them available as service through a SPARQL endpoint. The whole Fuseki setup allows us to query the datasets through rest endpoint.

The extensions are built as a jar file and this jar file is deployed to the Fuseki Web application. Through this, our extensions work as part of the web application through rest endpoint in addition to console-based application.

5.2 Implementation

Extensions for statistical functions are implemented using the support for extensions in SPARQL 1.1. In our system, we are adding the extensions to Apache Jena under two categories:

1. Custom Aggregates.
2. User-defined functions.

Using these extension mechanisms, we have implement statistical functions as a Java library.

5.3 Custom Aggregates

Custom aggregates allow us to define our own aggregates that work on grouping over a single field. Examples include finding a maximum value among the given attribute values, finding minimum, etc. Using SPARQL’s ability to implement custom aggregates, we implemented the below statistical functions:

1. Mean
2. Median
3. Standard Deviation
4. Variance

5. Skewness

Even though, the names of these statistical functions are intuitive, a brief description is provided on each of these functions.

Mean:

For a given dataset, the mean implies the average value. To be specific, it is the value given by sum of all the numbers in the dataset divided by the count of the values in the dataset. In the field of probability and statistics, mean and expected value can be used synonymously. The formula for mean is:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

Here μ denotes mean, n indicates the number of values. A summation of all the values divided by the count of the values gives the mean of a given variable in the dataset.

Median:

Median is the value that separates lower half from the higher half when the values are ordered in ascending or descending order. It is the middle value in the given dataset. Medians are helpful in understanding the distribution of data. This can be done by comparing mean and median values. By observing the difference between these values we can understand whether the data is left skewed or right skewed. The formula for median is:

$\text{Median} = ((n+1)/2)$th number in the series where the numbers are ordered. Here, n denotes the number of values for the given variable.

Standard Deviation:

Standard deviation signifies the amount of deviation of values from the mean of the dataset. It indicates how the data is distributed in a given dataset. A low value of standard deviation implies that the values are close to the mean and a high value implies that the values are farther from the mean of the dataset. The formula for standard deviation is:

$$s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$$

Here s denotes the standard deviation, n gives the number of values, x is the data value and $\bar{x}$ indicates the mean of the variable in the dataset.

Variance:

Variance is the expectation of the deviation from the mean of the dataset. It measures how far a value can go from the mean. Mathematically, the variance is the square of the standard deviation.

$$\nu = s^2$$

Here v denotes the variance and s denotes the standard deviation of the values.

Skewness:

Skewness is a measure of the asymmetry of distribution of data around the mean of the dataset. Negative skewness value indicates that more values lie towards left side of the mean value. A positive skewness indicates that more values lie towards the right side of the mean value.

5.4 Custom Aggregates Implementation:

For writing custom aggregates, we use the Accumulator and AccumulatorFactory interfaces in Apache Jena to define the rules for accumulating the values over a given predicate.

Accumulator

We need to implement the Accumulator interface to define our own logic for accumulating the object literals. For this purpose, the following methods of Accumulator interface should be implemented:

1. void accumulate (Binding binding, FunctionEnv functionEnv): The accumulate method is used to get the bound value to the given variable in the query. It tests whether a variable is bound to some object and if it is bound, we get the value. This value is added to a required datastructure (like list) that helps us in preserving the value for further calculation.
2. NodeValue getValue(): The getValue method helps us in retrieving the final result as RDF NodeValue. Inside this method we can add the logic for calculating the required statistical function over all the values that are accumulated in the accumulate method. It is here that we call the methods of R classes to calculate the desired R statistical function.

Below is the sample code that shows the logic to create an accumulator.

```
static class MeanAccumulator implements Accumulator{
   double meanValue = 0;
   private AggCustom aggCustom;

   MeanAccumulator(AggCustom aggCustom){
      this.aggCustom = aggCustom;
   }

   @Override
   public void accumulate(Binding binding, FunctionEnv functionEnv) {
      ExprList exprList = aggCustom.getExprList();
```

```
        for (Expr expr :
                exprList) {
            try{
                NodeValue nv = expr.eval(binding, functionEnv);
                if(nv.isLiteral()){
                    arrayList.add(Double.parseDouble(nv.asUnquotedString()));
                }
            }catch (ExprEvalException ex) {
                ex.printStackTrace();
            }
        }


    }
    @Override
    public NodeValue getValue() {
        Mean mean = new Mean();
        meanValue = mean.getMean(arrayList);
        DecimalFormat decimalFormat = new DecimalFormat("#.#####");
        double result = Double.parseDouble(decimalFormat.format(meanValue));
        return NodeValue.makeDecimal(result);
    }
}
```

Below is the sample code that calculates the mean values using the R engine.

```
public double getMean(ArrayList<Double> list){
    Rengine rengine = Rengine.getMainEngine();
    if(rengine == null){
        rengine = new Rengine(new String[] {"--no-save"},false,null);
    }
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("c(");
    for(int i = 0;i<list.size();i++){
        stringBuilder.append(Double.toString(list.get(i)));
        if(i<list.size()-1)
            stringBuilder.append(",");
     }
    stringBuilder.append(")");

    String jVector = stringBuilder.toString();
    rengine.eval("rVector="+jVector);
    rengine.eval("rMean=mean(rVector,na.rm=TRUE)");
    double mean = rengine.eval("rMean").asDouble();
    if(rengine != null)
        rengine.end();
    return mean;
 }
```

AccumulatorFactory

AccumulatorFactory is a factory to create accumulators. Once we create our own accumulator that implements the Accumulator interface, we need to instantiate our accumulator using the createAccumulator method of the AccumulatorFactory interface. Below is the code that instantiates the implemented accumulator.

```
 static AccumulatorFactory myAccumulatorFactory = new AccumulatorFactory() {

    @Override
    public Accumulator createAccumulator(AggCustom agg, boolean distinct) {
        return new MeanAccumulator(agg);
    }
 };
```

Once we have the factory, we register the factory along with a URI that identifies our custom aggregate method with the AggregateRegistry class. This AggregateRegistry class is a single global registry of custom aggregates.

```
String aggUri = "http://example.org/function#mean" ;
AggregateRegistry.register(aggUri,myAccumulatorFactory);
```

Once we have the URI for our user defined aggregate available in the AggregateRegistry, we can call the method using this URI. This can be used as part of the SPARQL queries.

All the statistical functions that rely on aggregation can be implemented using Custom Aggregators. There are other set of statistical functions that do not need aggregation but take in multiple inputs to give us a result. Such functions are implemented using user-defined Functions in Apache Jena.

5.5 User-Defined Functions

User defined functions allow us to write our own functions that work along with other SPARQL functions. For implementing user defined functions, we can either implement the Function interface or extend one of the base classes among FunctionBase, FunctionBase0, FunctionBase1, FunctionBase2, FunctionBase3, and FunctionBase4 depending on the number of arguments that our function takes.

The following set of statistical functions are identified that can be implemented using SPARQL 1.1 extensions for user defined functions.

1. Correlation functions:

   Pearson correlation,

   Kendall correlation, and

   Spearman correlation.

2. Test statistic functions

   One Sample T- test,

   Two Sample T-test,

   One Sample Z-test, and

   Two Sample Z-test.

Correlation:

Correlation is a concept that helps us in understanding the relationship between two different variables in the given dataset. This measure helps us to understand if there is a directly or indirectly proportional relationship and also signifies the strength of the relation. Correlation helps us to compare disparate variables in datasets and understand the relationship between them.

Correlation coefficient is a number that signifies the type of correlation or dependence between variables. It gives a measure of the strength and direction of the linear relationship between two variables. The value of the coefficient lies between +1 and -1.

There are three types of correlation coefficients:

1. Positive correlation: A positive coefficient indicates that as the value of one variable increases, so does the value of the second variable.

2. Negative correlation: A negative value indicates that as the value of one variable increases, there is a decrease in the value of the second variable.

3. None: There is no relationship between the values of the two variables. The second variable is unaffected by the increase or decrease of the first variable. Such values are close to 0.

In statistics, we primarily consider or measure three types of correlations:

1. Pearson Correlation,
2. Spearman Correlation, and
3. Kendall Correlation.

Pearson Correlation: This is one of the most widely used correlations to measure the relationship between variables that are linearly related. Also, the Pearson Correlation is unaffected by the linear transformations in the variable values. The value of the coefficient is interpreted in different ways depending on the type of dataset. For example, in social-sciences, the values near 1, such as 0.8, 0.9, are considered as high linear correlation values. While measuring physical laws a value of 0.8 may not be considered good enough to state a high linear correlation.

Spearman Correlation: Unlike the Pearson Correlation which considers the data values, the Spearman Correlation considers the rank of the data. Ranking is done by putting the values of a given variable in order and assigning the numbers to the order. It is a non-parametric test and the results are distribution free. Instead of measuring a linear relationship, the Spearman Correlation helps us in understanding monotonic relations. In Spearman Correlation, the calculations are based on the deviations in the data. This test is sensitive to error and discrepancies in the data.

Kendall Correlation: Similar to the Spearman Correlation, the Kendall Correlation also considers the ranks of the data. A positive Kendall coefficient signifies that the ranks of both variables are increasing whereas negative value signifies that the ranks are inversely related. With the Kendall Correlation, calculations are based on concordant and discordant pairs of data. Concordant and discordant pairs indicate the distribution of ranks in a dataset. A concordant pair is one where a subject or a statistic ranked higher on one observation also ranks higher on the second observation. A discordant pair is one where a statistic ranking is higher on one observation ranks lower on the second observation.

Test Statistics

Test statistics are used in scenarios where we do hypothesis testing. Such statistics help us in deciding whether to accept or reject a null hypothesis. Hypothesis testing allows us to test if our assumption based on the observations is true or not. Hypothesis testing is a way to figure out if results from a test are valid or repeatable. For example, if someone said they had found a new drug that cures cancer, you would want to be sure that it is true. A hypothesis test will tell you if it is probably true, or probably not true.

While performing a hypothesis test, a dataset with distribution like t-distribution or a normal distribution is used. The t-test and the z-test compare statistical measures like means of the given inputs and check if they are same or different and allow us to conclude on our hypothesis. Also, such tests give us an idea on how significant the difference between the statistical measures is.

An example scenario:

A drug company wants to test a new cancer drug. The company tests some people by splitting them into two groups. One group is given a placebo pill and the second group is given the new drug. The company then checks the life expectancy of two groups. It is noticed that the first group shows a life expectancy increased by 5 years while second group shows an increase by 6 years. Then it is observed that the drug works. For testing this, test statistics namely t-test or z-test are applied to determine if the same experiment is repeatable for entire population and the results are as expected. (Example taken from http://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/)

From this scenario, we propose two hypotheses.

1. Null Hypothesis: The drug has no effect on the life expectancy. This stated in terms of statistics is, the mean life expectancy of the group taking the drug is equal to the mean life expectancy of the group that does not consume the drug.

H_0: $\mu_1 = \mu_2$

2. Alternative Hypothesis: The drug has an effect on the life expectancy. This stated in terms of statistics is, the mean life expectancy of the group taking the drug is not equal to the mean life expectancy of the group that doesn’t take the drug.

H_a: $\mu_1 \neq \mu_2$

Our testing should give us an answer if the null hypothesis is true or the alternative hypothesis is true. For each of these hypotheses, we test the samples we take and obtain probabilities under each category to support our results.

Below is a description on the test statistic functions.

Z Test: One of the common measures to test a hypothesis is Z test. A Z test is conducted when the sample size is large and the population standard deviation or variance is known. Also, the dataset should approximately follow a normal distribution.

To check whether the dataset follows a normal distribution, we can use a histogram plot or Q-Q plot (Quantile-Qunatile plot). The histogram should imitate a bell curve if the data follows a normal distribution. In our project, we accept some skewness in the distribution of data and we leave it to the statisticians to normalize the data before conducting the tests.

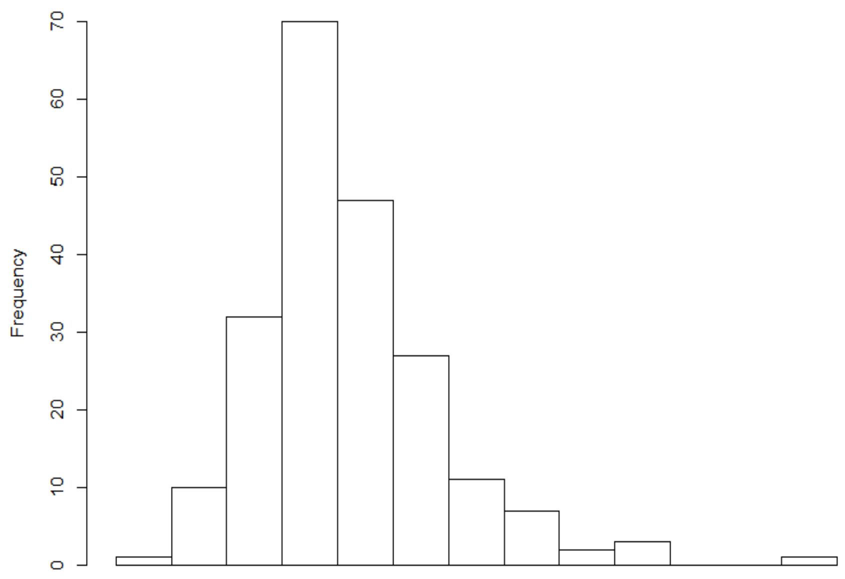


Figure 2: Histogram distribution of a sample data. This indicates that the data is slightly skewed towards the right. However, we consider the data to follow a near normal distribution.

We read the confidence level (c) or significance level (α) and determine the Z value from the Z scores table. We accept or reject the null hypothesis based on whether the calculated Z score is less than or greater than the required value. Also, the conditions are based on whether we are conducting an upper tailed or lower tailed or two-tailed Z test. For an upper tailed test, if the calculated z score is greater than the Z score from the table (http://www.statisticshowto.com/tables/z-table/), we reject the null hypothesis. For a lower tailed test, if the calculated Z score is greater than the Z score from the table, we reject the null hypothesis. For a two-tailed test, if the calculated value is greater than the positive Z score or less than the negative z scores from the table, we reject the null hypothesis.

Also, we conduct the Z test with a single sample or two samples to understand the differences in a single population or between two populations.

T Test: A T test is used for hypothesis testing if our data has certain constraints. The constraints are the sample size should be small, i.e., less than 30 and the population standard distribution or variance is unknown. Also, the data follows a T-distribution. A T-distribution curve is highly similar to normal distribution curve with the area under curve being slightly fatter and shorter. This implies that the tails are fatter.

The analysis based on the results of T score is similar to that of Z score but we use a T table (http://www.statisticshowto.com/tables/t-distribution-table/) to fix our threshold values. Instead of using just significance level, we use the degrees of freedom (sample size minus one) to read and determine the required T scores from the T table.

5.6 User-Defined Functions Implementation

Unlike custom aggregates, for implementing correlations and test statistics we use extension functions. Apache Jena provides interfaces and abstract classes to implement or override to realize our own custom function evaluation. The classes that we can extend are FunctionBase, FunctionBase0, FunctionBase1, FunctionBase2, FunctionBase3, and FunctionBase4. The suffix digit indicates the number of arguments that the function takes. If we have more than 4 arguments to pass in, we can always extend the FunctionBase class and define our own extension class.

The FunctionBase family of classes has two methods that can be overridden and one abstract method that can be implemented.

1. checkBuild(String uri, ExprList args): checks whether the number of arguments is as expected. Otherwise it throws a QueryBuildException.

2. Exec(List<NodeValue> args): checks the number of arguments and converts each argument as a separate NodeValue argument to the abstract function exec();
3. Abstract NodeValue exec(NodeValue v1, NodeValue v2……): This method can be overridden by the extending class in order to evaluate the function expressions in a desired way by the user.

Below is the sample code for implementing Pearson Correlation.

```
public class PearsonCorrelation extends FunctionBase2{
   static {
FunctionRegistry.get().put("http://example.org/function#pearsoncorr",PearsonCorrelati
on.class);
   }

   public static void init(){
FunctionRegistry.get().put("http://example.org/function#pearsoncorr",PearsonCorrelati
on.class);
   }

   public double getPearsonCorrelation(String input1, String input2){
      Correlation correlation = new Correlation();
      return correlation.getPearsonCorrelation(input1,input2);
   }

   @Override
   public NodeValue exec(NodeValue v1, NodeValue v2) {
      double pearsonCorrelationVal =
getPearsonCorrelation(v1.asUnquotedString(),v2.asUnquotedString());
      DecimalFormat decimalFormat = new DecimalFormat("#.#####");
```

```
        double result =
Double.parseDouble(decimalFormat.format(pearsonCorrelationVal));
        return NodeValue.makeDecimal(result);
    }
}
```

This function tests and retrieves the strength of correlation between two variables. So, we pass in two variables to the functions and retrieve the correlation value. In the exec method, we are getting the variables bound to data. This data is in turn sent to the R-Integration module for evaluating results.

Below is the R code for computing Pearson Correlation value.

```
public double getPearsonCorrelation(String input1, String input2){
    double correlationValue =0;
    StringBuilder stringBuilderInput1 = new StringBuilder();
    StringBuilder stringBuilderInput2 = new StringBuilder();
    String[] inputArr1 = input1.split(",");
    String[] inputArr2 = input2.split(",");
    int len = 0;
    if(inputArr1.length < inputArr2.length)
        len = inputArr1.length;
    else
        len = inputArr2.length;

    String[] inputArr1Dst = Arrays.copyOf(inputArr1,len);
    String[] inputArr2Dst = Arrays.copyOf(inputArr2,len);

    input1 =  Arrays.toString(inputArr1Dst);
    input2 = Arrays.toString(inputArr2Dst);
    input1 = input1.substring(1,input1.length()-1);
    input2 = input2.substring(1, input2.length()-1);
    stringBuilderInput1.append("c(");
    stringBuilderInput2.append("c(");
    stringBuilderInput1.append(input1);
    stringBuilderInput2.append(input2);
    stringBuilderInput1.append(")");
    stringBuilderInput2.append(")");

    Rengine rengine = Rengine.getMainEngine();
    if(rengine == null){
        rengine = new Rengine(new String[] {"--no-save"},false,null);
    }
    rengine.eval("rXVector=as.numeric("+stringBuilderInput1.toString()+")");
    rengine.eval("rYVector=as.numeric("+stringBuilderInput2.toString()+")");
    rengine.eval("result=cor(rXVector,rYVector,method=c('pearson'))");
    correlationValue = rengine.eval("result").asDouble();

    if(rengine!= null)
        rengine.end();
    return correlationValue;
}
```

We construct the R vectors from the given strings in Java and evaluate the correlation function in R. The result is returned as a double value. This result is again constructed as a NodeValue in the Extension module and the final result is sent to the user. All functions that utilize Jena’s user-defined function extension mechanism are implemented in a similar fashion.

CHAPTER 6

EVALUATION

The goal of our library is to allow users to execute statistical queries on RDF data. This still requires users to have a basic understanding of writing SPARQL queries and using functions. For demonstrating the usage and evaluating the queries, we used Apache Jena Fuseki endpoint, which acts a SPARQL endpoint for the RDF datasets we load into the triple store. Also, the same functions are executed in R by exporting the dataset to R. This is to check if the results retrieved from SPARQL and R are the same or not. We either export the sample dataset to R or use RRDF and SPARQL libraries in R to query directly against the Fuseki endpoint.

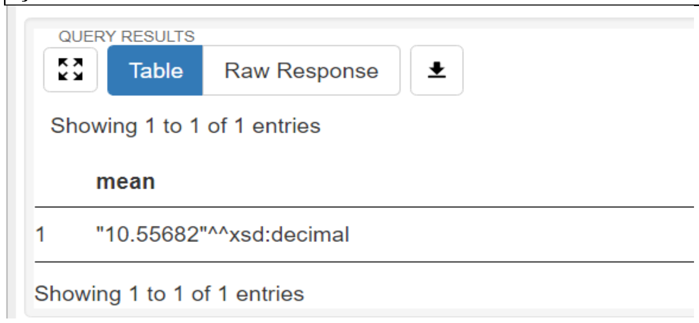
The datasets are from Data.gov (https://www.data.gov/) and include both federal and state datasets. These datasets are provided by the US government as part of the Open Data initiative. Citizens can leverage this open data in various studies to understand the hidden implications. Also, the site acts as a rich resource for data scientists, tech entrepreneurs, and developers to conduct various studies. Even though most of the data is represented in the form of csv, json and xml, we have a good number of datasets represented in RDF to perform evaluation for this work. We have loaded the datasets into Apache Jena Fuseki endpoint and executed test queries.

6.1 Mean

For evaluating the mean functionality, we utilized Behavioral Risk Factor data (https://catalog.data.gov/dataset/behavioral-risk-factor-data-heart-disease-amp-stroke-prevention) from Data.gov and below are the queries with test results.

Query 1: To get the average value of "Prevalence of major cardiovascular diseases among adults in the US".

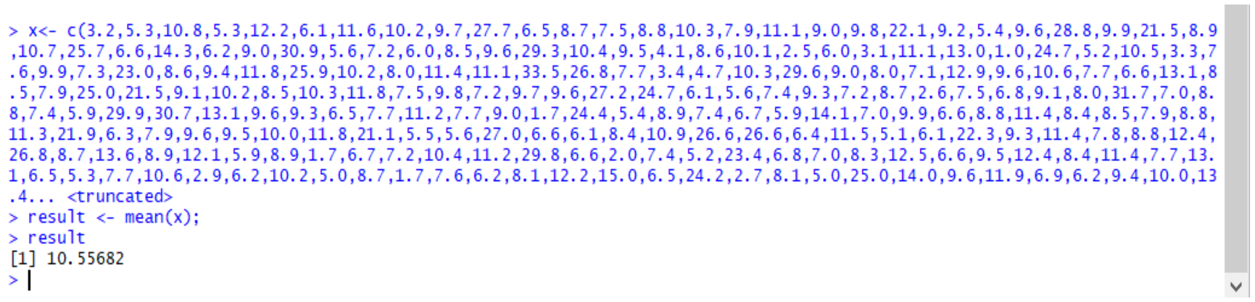
```
PREFIX fn:<http://example.org/function#>
SELECT (fn:mean(?o) AS ?mean)
WHERE {
  ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> "Prevalence of
major cardiovascular disease among US adults (18+); BRFSS" .
  ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?o .
  FILTER(regex(?o,'[0-9.]'))
}
```



```
Showing 1 to 1 of 1 entries

     mean
1    "10.55682"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 3: Query result from Apache Jena Fuseki endpoint for mean of "Prevalence of major cardiovascular diseases" in Behavioral Risk factors dataset.

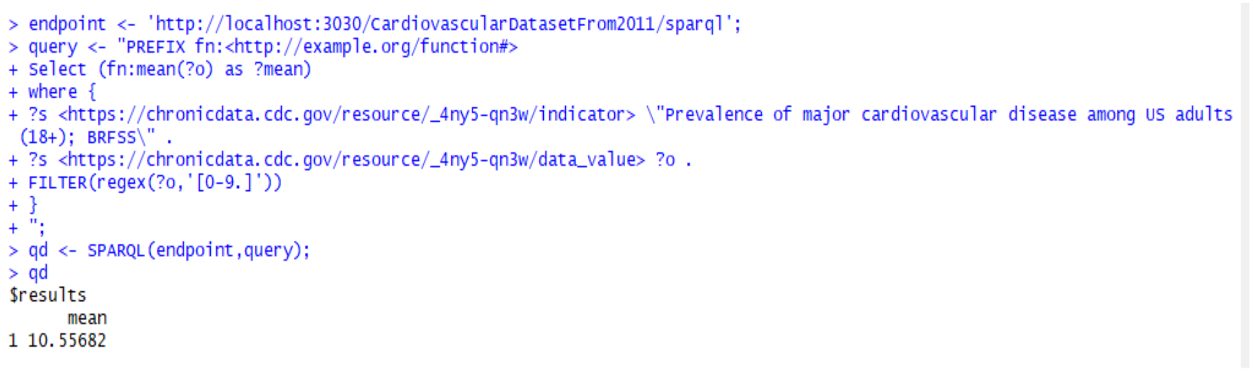
In this query, `fn:mean` indicates that this is a function from our library. `fn:mean` function aggregates the results that are bound to the variable ?o, and gets the average value as output. When the data was exported and tested in R, we got the same result 10.55682, as shown in below figure. For testing in R, we created a vector with all data values for the given constraints.



```
> x<- c(3.2,5.3,10.8,5.3,12.2,6.1,11.6,10.2,9.7,27.7,6.5,8.7,7.5,8.8,10.3,7.9,11.1,9.0,9.8,22.1,9.2,5.4,9.6,28.8,9.9,21.5,8.9
,10.7,25.7,6.6,14.3,6.2,9.0,30.9,5.6,7.2,6.0,8.5,9.6,29.3,10.4,9.5,4.1,8.6,10.1,2.5,6.0,3.1,11.1,13.0,1.0,24.7,5.2,10.5,3.3,7
.6,9.9,7.3,23.0,8.6,9.4,11.8,25.9,10.2,8.0,11.4,11.1,33.5,26.8,7.7,3.4,4.7,10.3,29.6,9.0,8.0,7.1,12.9,9.6,10.6,7.7,6.6,13.1,8
.5,7.9,25.0,21.5,9.1,10.2,8.5,10.3,11.8,7.5,9.8,7.2,9.7,9.6,27.2,24.7,6.1,5.6,7.4,9.3,7.2,8.7,2.6,7.5,6.8,9.1,8.0,31.7,7.0,8.
8,7.4,5.9,29.9,30.7,13.1,9.6,9.3,6.5,7.7,11.2,7.7,9.0,1.7,24.4,5.4,8.9,7.4,6.7,5.9,14.1,7.0,9.9,6.6,8.8,11.4,8.4,8.5,7.9,8.8,
11.3,21.9,6.3,7.9,9.6,9.5,10.0,11.8,21.1,5.5,5.6,27.0,6.6,6.1,8.4,10.9,26.6,26.6,6.4,11.5,5.1,6.1,22.3,9.3,11.4,7.8,8.8,12.4,
26.8,8.7,13.6,8.9,12.1,5.9,8.9,1.7,6.7,7.2,10.4,11.2,29.8,6.6,2.0,7.4,5.2,23.4,6.8,7.0,8.3,12.5,6.6,9.5,12.4,8.4,11.4,7.7,13.
1,6.5,5.3,7.7,10.6,2.9,6.2,10.2,5.0,8.7,1.7,7.6,6.2,8.1,12.2,15.0,6.5,24.2,2.7,8.1,5.0,25.0,14.0,9.6,11.9,6.9,6.2,9.4,10.0,13
.4... <truncated>
> result <- mean(x);
> result
[1] 10.55682
> |
```

Figure 4: SPARQL query in R for mean where a vector of data values is given as input.

Also, we repeated the same experiment using SPARQL package in R. SPARQL package allows us to query the endpoint directly. The resultant mean value is the same as in the previous cases.



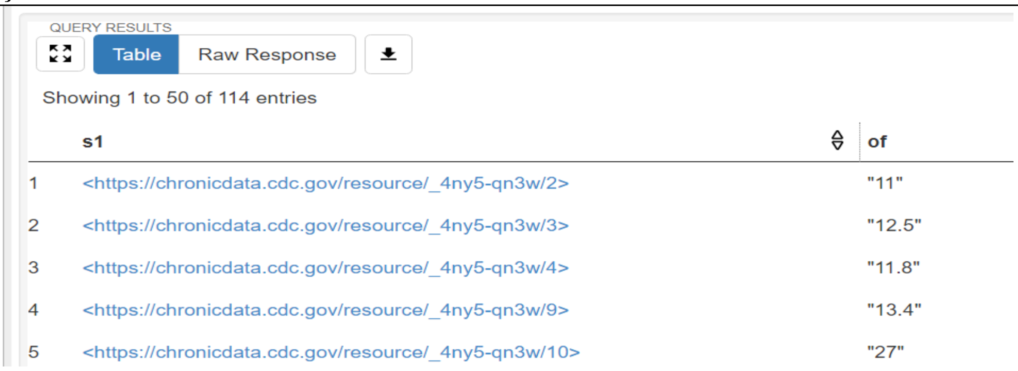
```
> endpoint <- 'http://localhost:3030/CardiovascularDatasetFrom2011/sparql';
> query <- "PREFIX fn:<http://example.org/function#>
+ Select (fn:mean(?o) as ?mean)
+ where {
+ ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> \"Prevalence of major cardiovascular disease among US adults
 (18+); BRFSS\" .
+ ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?o .
+ FILTER(regex(?o,'[0-9.]'))
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
      mean
1 10.55682
```

Figure 5: Result in R when mean is queried through SPARQL package in R.

Query 2: Retrieve a list of all subjects with a data value greater than the mean value of “Prevalence of major cardiovascular diseases”.

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s1 ?of
WHERE{
  ?s1 <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> "Prevalence of
major cardiovascular disease among US adults (18+); BRFSS" .
  ?s1 <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?of .
            {
            SELECT (fn:mean(?o) AS ?mean)
            WHERE
```

```
          {?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> "Prevalence
of major cardiovascular disease among US adults (18+); BRFSS" .
          ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?o .
          FILTER(regex(?o,'[0-9]')) .
   }
}
  FILTER (xsd:decimal(?of)  > ?mean)
}
```



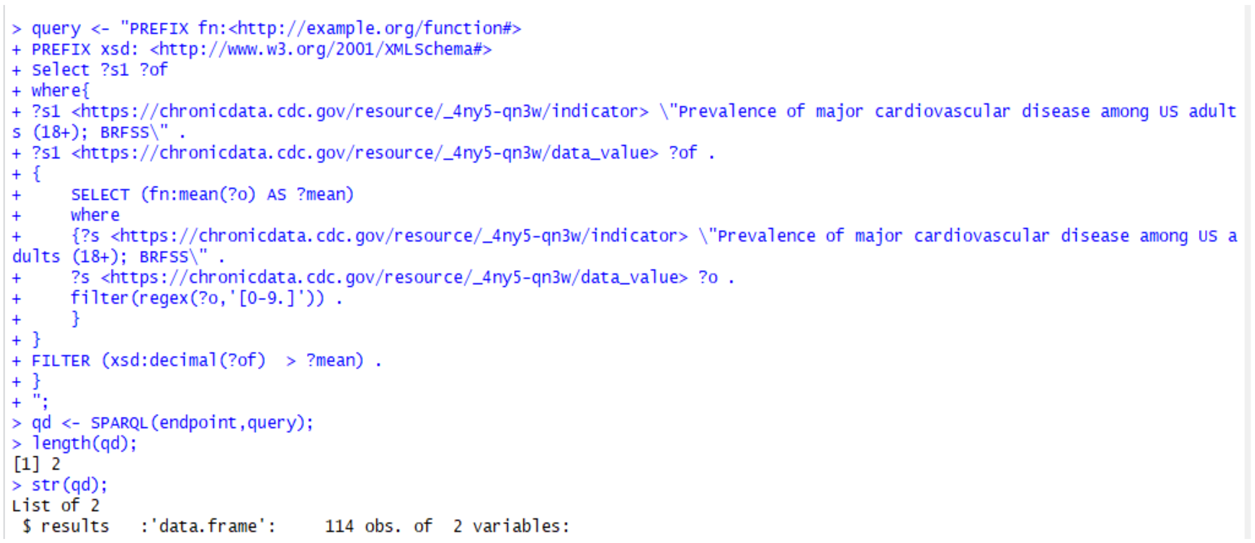
Showing 1 to 50 of 114 entries

	s1	of
1	<https://chronicdata.cdc.gov/resource/_4ny5-qn3w/2>	"11"
2	<https://chronicdata.cdc.gov/resource/_4ny5-qn3w/3>	"12.5"
3	<https://chronicdata.cdc.gov/resource/_4ny5-qn3w/4>	"11.8"
4	<https://chronicdata.cdc.gov/resource/_4ny5-qn3w/9>	"13.4"
5	<https://chronicdata.cdc.gov/resource/_4ny5-qn3w/10>	"27"

Figure 6: Query result in Apache Jena Fuseki endpoint for retrieving a list of subjects with data value greater than mean value.

We got 114 subjects that have their data value greater than the mean value of “Prevalence of major cardiovascular disease among US adults”.

We repeated the same experiment in R using SPARQL package and below are the results.



```
> query <- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ Select ?s1 ?of
+ where{
+ ?s1 <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> \"Prevalence of major cardiovascular disease among US adult
s (18+); BRFSS\" .
+ ?s1 <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?of .
+ {
+     SELECT (fn:mean(?o) AS ?mean)
+     where
+     {?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/indicator> \"Prevalence of major cardiovascular disease among US a
dults (18+); BRFSS\" .
+     ?s <https://chronicdata.cdc.gov/resource/_4ny5-qn3w/data_value> ?o .
+     filter(regex(?o,'[0-9.]')) .
+     }
+ }
+ FILTER (xsd:decimal(?of)  > ?mean) .
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> length(qd);
[1] 2
> str(qd);
List of 2
 $ results    :'data.frame':     114 obs. of  2 variables:
```

Figure 7: Results in R for data value greater than mean value.

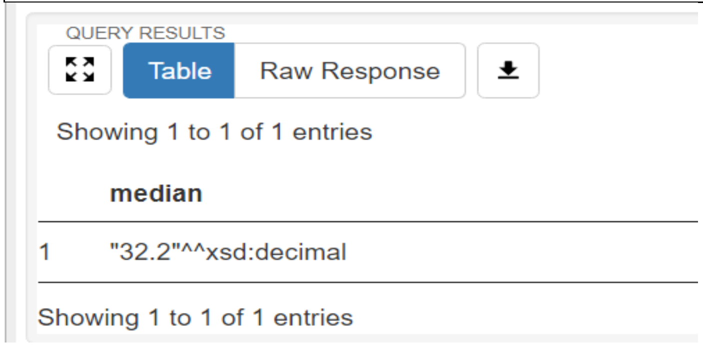
These results indicate that there are 114 rows that come as output from the above query, and this is the same as the results in Fuseki endpoint.

6.2 Median

For demonstrating the results of the median, we used a dataset that contains information about Chronic Diseases. (https://catalog.data.gov/dataset/u-s-chronic-disease-indicators-cdi-e50c9).

Query 3: To get the median of all data values associated with "Alcohol usage among youth".

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:median(?o) AS ?median)
WHERE{
?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
 ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
  FILTER(regex(?o,'[0-9.]'))
}
```



```
Showing 1 to 1 of 1 entries

        median
1       "32.2"^^xsd:decimal
Showing 1 to 1 of 1 entries
```

Figure 8: Query result in Apache Jena Fuseki endpoint for median of alcohol usage among youth in Chronic Diseases dataset.

When the data was exported and tested in R, the result was 32.2 and this is shown in the figure below.

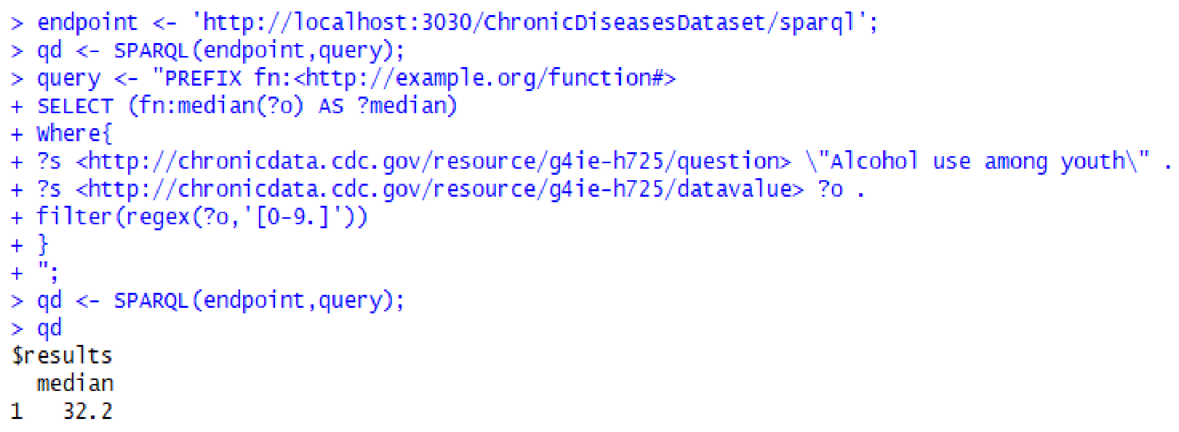
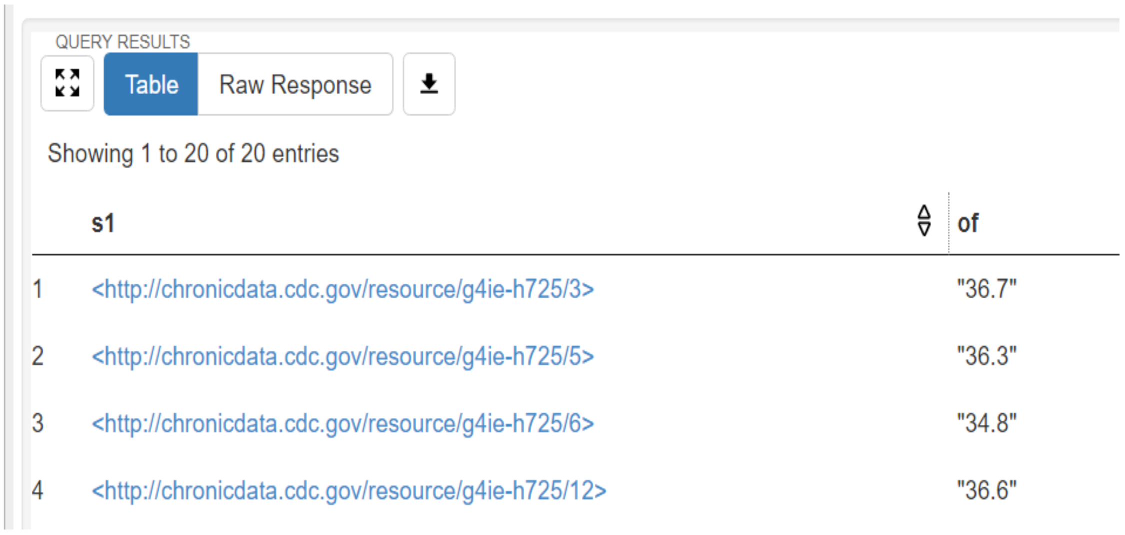

```
> endpoint <- 'http://localhost:3030/ChronicDiseasesDataset/sparql';
> qd <- SPARQL(endpoint,query);
> query <- "PREFIX fn:<http://example.org/function#>
+ SELECT (fn:median(?o) AS ?median)
+ where{
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+ filter(regex(?o,'[0-9.]'))
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
  median
1   32.2
```

Figure 9: SPARQL query executed in R for retrieving median value.

Query 4: To retrieve a list of all subjects that lie to the right of the median value.

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s1 ?of
WHERE{
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .
            {
            SELECT (fn:median(?o) AS ?median)
            WHERE{
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
             FILTER(regex(?o,'[0-9.]')) .
            }
  }
  FILTER(regex(?of,'[0-9.]')) .
  FILTER (xsd:decimal(?of)  > ?median)
}
```

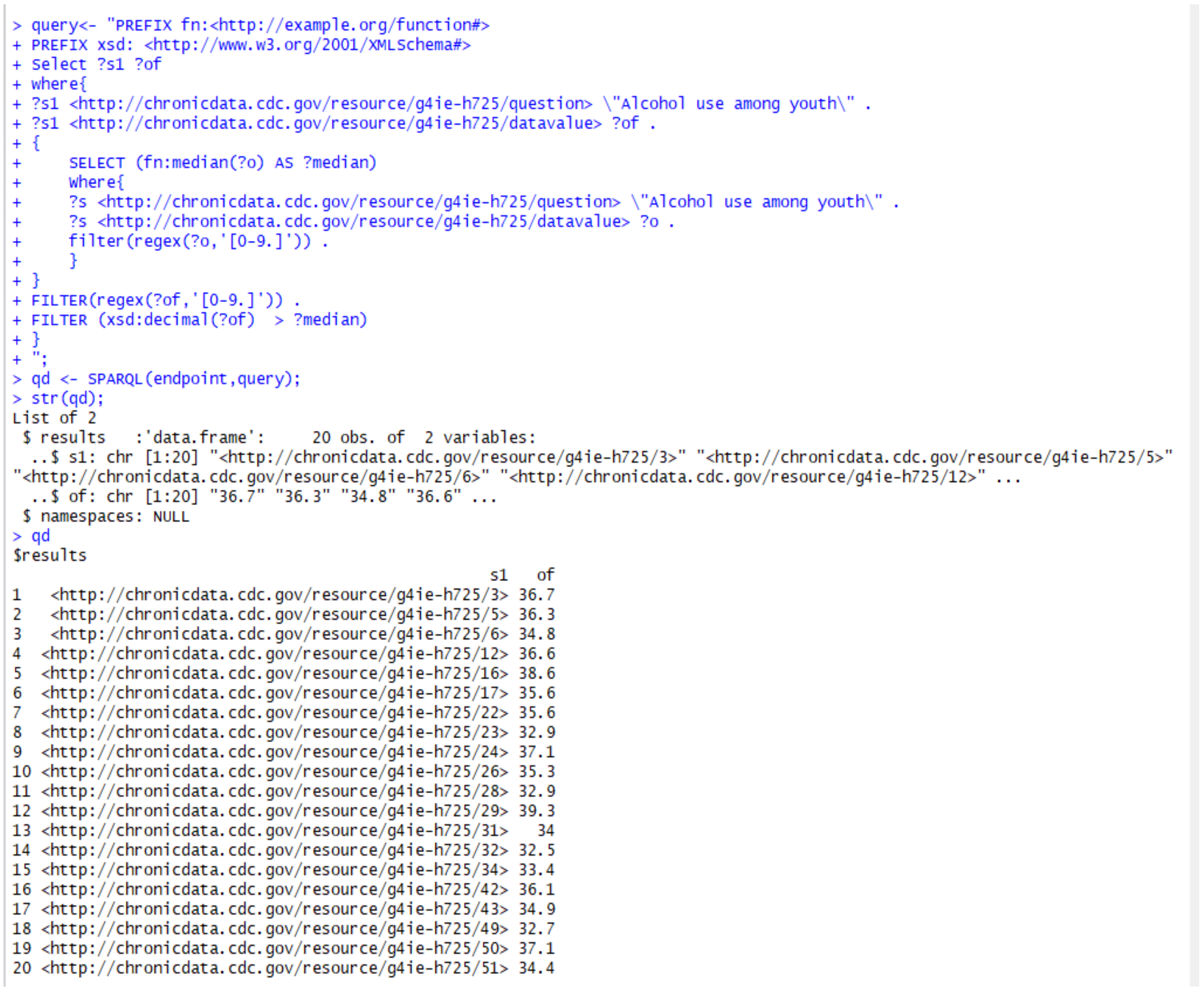


Showing 1 to 20 of 20 entries

	s1	of
1	<http://chronicdata.cdc.gov/resource/g4ie-h725/3>	"36.7"
2	<http://chronicdata.cdc.gov/resource/g4ie-h725/5>	"36.3"
3	<http://chronicdata.cdc.gov/resource/g4ie-h725/6>	"34.8"
4	<http://chronicdata.cdc.gov/resource/g4ie-h725/12>	"36.6"

Figure 10: Query result in Apache Jena Fuseki endpoint for data values which are greater than the median.

This indicates that there are 20 subjects with a data value greater than the median value. The results show that there are 20 such subjects which have average alcohol usage above the median value. The same experiment is repeated in R using SPARQL package.



```
> query<- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ Select ?s1 ?of
+ where{
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .
+ {
+      SELECT (fn:median(?o) AS ?median)
+      where{
+      ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+      ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+      filter(regex(?o,'[0-9.]')) .
+      }
+ }
+ FILTER(regex(?of,'[0-9.]')) .
+ FILTER (xsd:decimal(?of)  > ?median)
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> str(qd);
List of 2
 $ results    :'data.frame':     20 obs. of  2 variables:
  ..$ s1: chr [1:20] "<http://chronicdata.cdc.gov/resource/g4ie-h725/3>" "<http://chronicdata.cdc.gov/resource/g4ie-h725/5>"
"<http://chronicdata.cdc.gov/resource/g4ie-h725/6>" "<http://chronicdata.cdc.gov/resource/g4ie-h725/12>" ...
  ..$ of: chr [1:20] "36.7" "36.3" "34.8" "36.6" ...
 $ namespaces: NULL
> qd
$results
                                                    s1   of
1    <http://chronicdata.cdc.gov/resource/g4ie-h725/3> 36.7
2    <http://chronicdata.cdc.gov/resource/g4ie-h725/5> 36.3
3    <http://chronicdata.cdc.gov/resource/g4ie-h725/6> 34.8
4   <http://chronicdata.cdc.gov/resource/g4ie-h725/12> 36.6
5   <http://chronicdata.cdc.gov/resource/g4ie-h725/16> 38.6
6   <http://chronicdata.cdc.gov/resource/g4ie-h725/17> 35.6
7   <http://chronicdata.cdc.gov/resource/g4ie-h725/22> 35.6
8   <http://chronicdata.cdc.gov/resource/g4ie-h725/23> 32.9
9   <http://chronicdata.cdc.gov/resource/g4ie-h725/24> 37.1
10  <http://chronicdata.cdc.gov/resource/g4ie-h725/26> 35.3
11  <http://chronicdata.cdc.gov/resource/g4ie-h725/28> 32.9
12  <http://chronicdata.cdc.gov/resource/g4ie-h725/29> 39.3
13  <http://chronicdata.cdc.gov/resource/g4ie-h725/31>   34
14  <http://chronicdata.cdc.gov/resource/g4ie-h725/32> 32.5
15  <http://chronicdata.cdc.gov/resource/g4ie-h725/34> 33.4
16  <http://chronicdata.cdc.gov/resource/g4ie-h725/42> 36.1
17  <http://chronicdata.cdc.gov/resource/g4ie-h725/43> 34.9
18  <http://chronicdata.cdc.gov/resource/g4ie-h725/49> 32.7
19  <http://chronicdata.cdc.gov/resource/g4ie-h725/50> 37.1
20  <http://chronicdata.cdc.gov/resource/g4ie-h725/51> 34.4
```

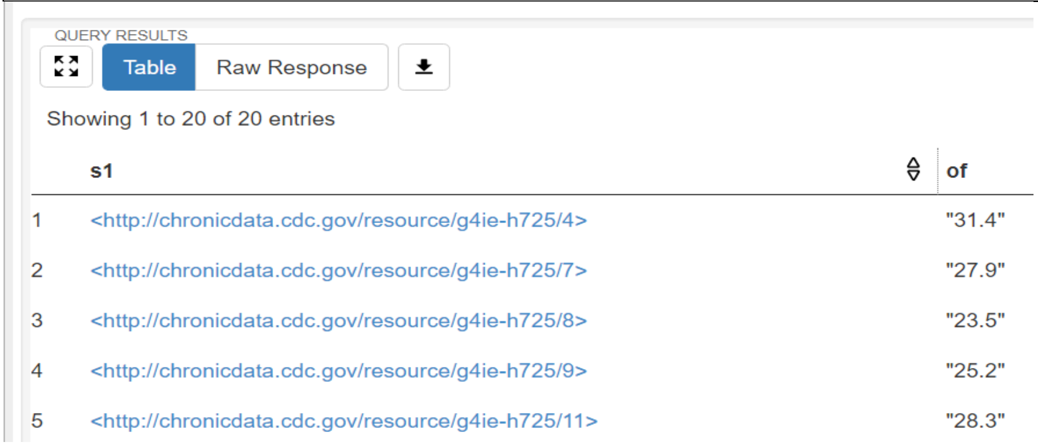
Figure 11: SPARQL query executed in R for data value greater than the median.

The result in R is the same as in Apache Jena Fuseki. The result set has 20 observations. The results in R also indicate that there are 20 subjects with a data value greater than the median.

Query 5: To get the list of all subjects that lie to the left of the median value.

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s1 ?of
WHERE{
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .
          {
```

```
          SELECT (fn:median(?o) AS ?median)
          WHERE{
          ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
          ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
           FILTER(regex(?o,'[0-9.]')) .
          }
  }
  FILTER(regex(?of,'[0-9.]')) .
  FILTER (xsd:decimal(?of)  < ?median)
}
```



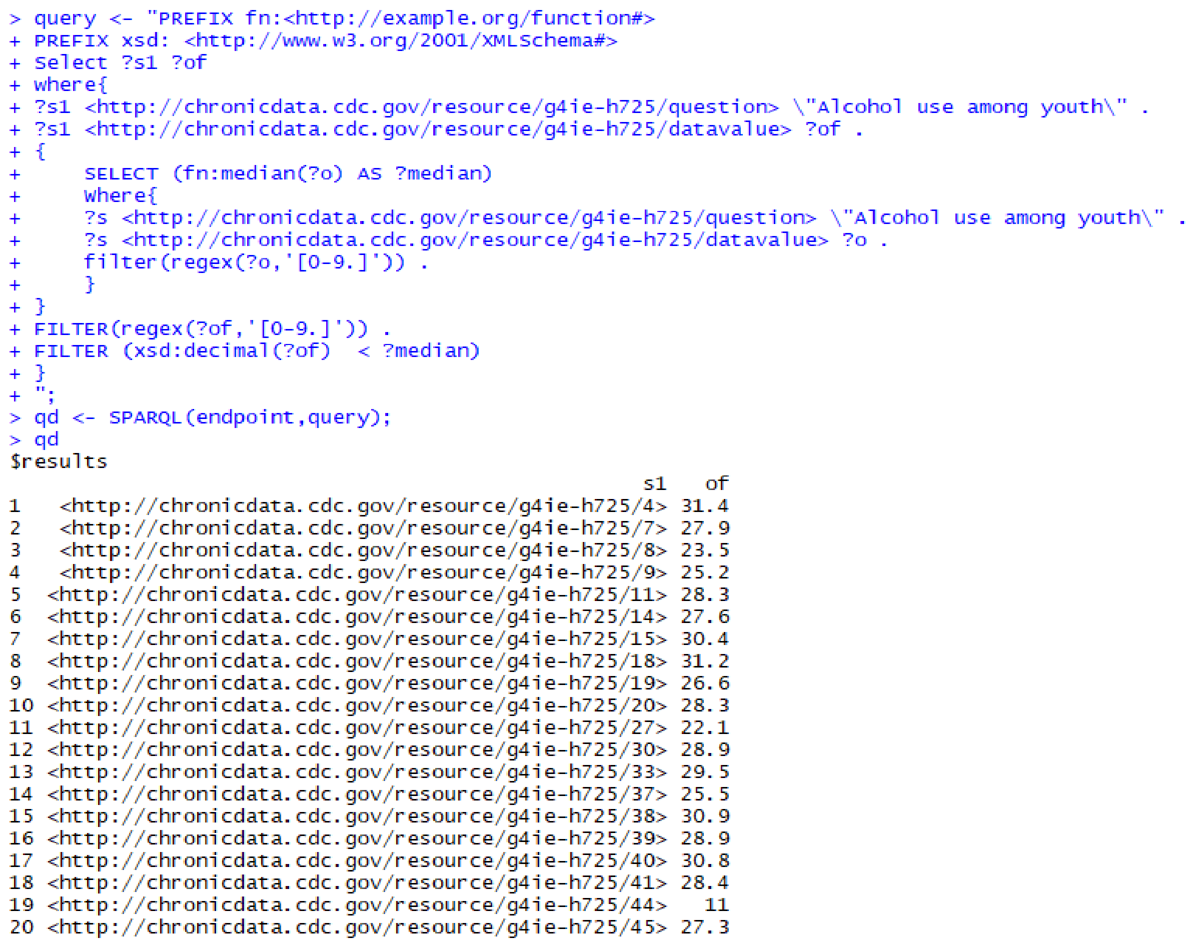
Showing 1 to 20 of 20 entries

	s1	of
1	<http://chronicdata.cdc.gov/resource/g4ie-h725/4>	"31.4"
2	<http://chronicdata.cdc.gov/resource/g4ie-h725/7>	"27.9"
3	<http://chronicdata.cdc.gov/resource/g4ie-h725/8>	"23.5"
4	<http://chronicdata.cdc.gov/resource/g4ie-h725/9>	"25.2"
5	<http://chronicdata.cdc.gov/resource/g4ie-h725/11>	"28.3"

Figure 12: Query result in Apache Jena Fuseki for data values lesser than the median.

This indicates that there are 20 subjects with a data value lesser than the median value.

The same experiment is repeated in R and the results are as shown in the figure below.



```
> query <- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ Select ?s1 ?of
+ where{
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .
+ {
+     SELECT (fn:median(?o) AS ?median)
+     where{
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+     filter(regex(?o,'[0-9.]')) .
+     }
+ }
+ FILTER(regex(?of,'[0-9.]')) .
+ FILTER (xsd:decimal(?of)  < ?median)
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
                                                     s1   of
1    <http://chronicdata.cdc.gov/resource/g4ie-h725/4> 31.4
2    <http://chronicdata.cdc.gov/resource/g4ie-h725/7> 27.9
3    <http://chronicdata.cdc.gov/resource/g4ie-h725/8> 23.5
4    <http://chronicdata.cdc.gov/resource/g4ie-h725/9> 25.2
5   <http://chronicdata.cdc.gov/resource/g4ie-h725/11> 28.3
6   <http://chronicdata.cdc.gov/resource/g4ie-h725/14> 27.6
7   <http://chronicdata.cdc.gov/resource/g4ie-h725/15> 30.4
8   <http://chronicdata.cdc.gov/resource/g4ie-h725/18> 31.2
9   <http://chronicdata.cdc.gov/resource/g4ie-h725/19> 26.6
10  <http://chronicdata.cdc.gov/resource/g4ie-h725/20> 28.3
11  <http://chronicdata.cdc.gov/resource/g4ie-h725/27> 22.1
12  <http://chronicdata.cdc.gov/resource/g4ie-h725/30> 28.9
13  <http://chronicdata.cdc.gov/resource/g4ie-h725/33> 29.5
14  <http://chronicdata.cdc.gov/resource/g4ie-h725/37> 25.5
15  <http://chronicdata.cdc.gov/resource/g4ie-h725/38> 30.9
16  <http://chronicdata.cdc.gov/resource/g4ie-h725/39> 28.9
17  <http://chronicdata.cdc.gov/resource/g4ie-h725/40> 30.8
18  <http://chronicdata.cdc.gov/resource/g4ie-h725/41> 28.4
19  <http://chronicdata.cdc.gov/resource/g4ie-h725/44>   11
20  <http://chronicdata.cdc.gov/resource/g4ie-h725/45> 27.3
```

Figure 13: SPARQL query executed in R for data values lesser than the median.

The result in R is the same as in Apache Jena Fuseki. The result set in R has 20 observations.

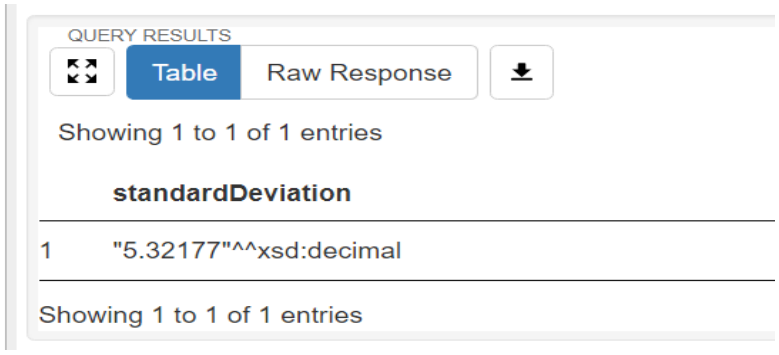
6.3 Standard Deviation

For testing standard deviation function, the dataset with information on Chronic diseases is used.

Query 6: To get the standard deviation in the data values related to “Alcohol usage among youth”.

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT (fn:sd(?o) AS ?standardDeviation)
WHERE{
```

```
?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
FILTER(regex(?o,'[0-9.]')) .
}
```



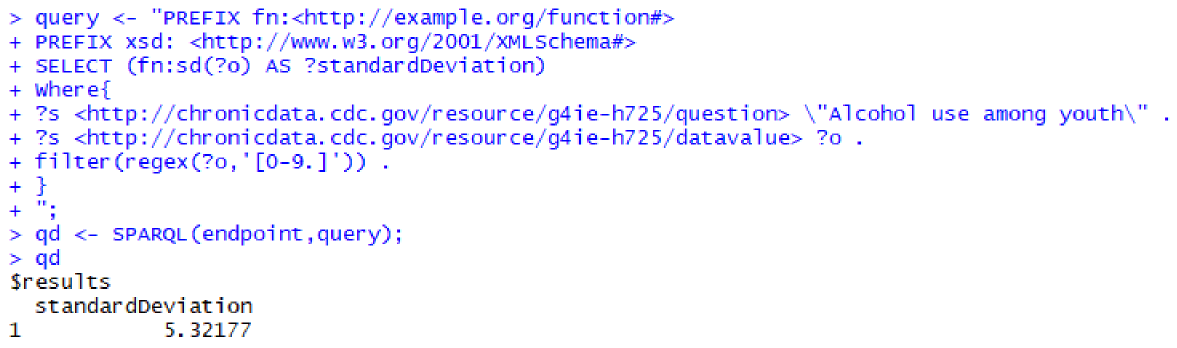
Showing 1 to 1 of 1 entries

	standardDeviation
1	"5.32177"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 14: Query result in Apache Jena Fuseki for standard deviation.

When the data was loaded in R, it gave the same resultant standard deviation as 5.32177.

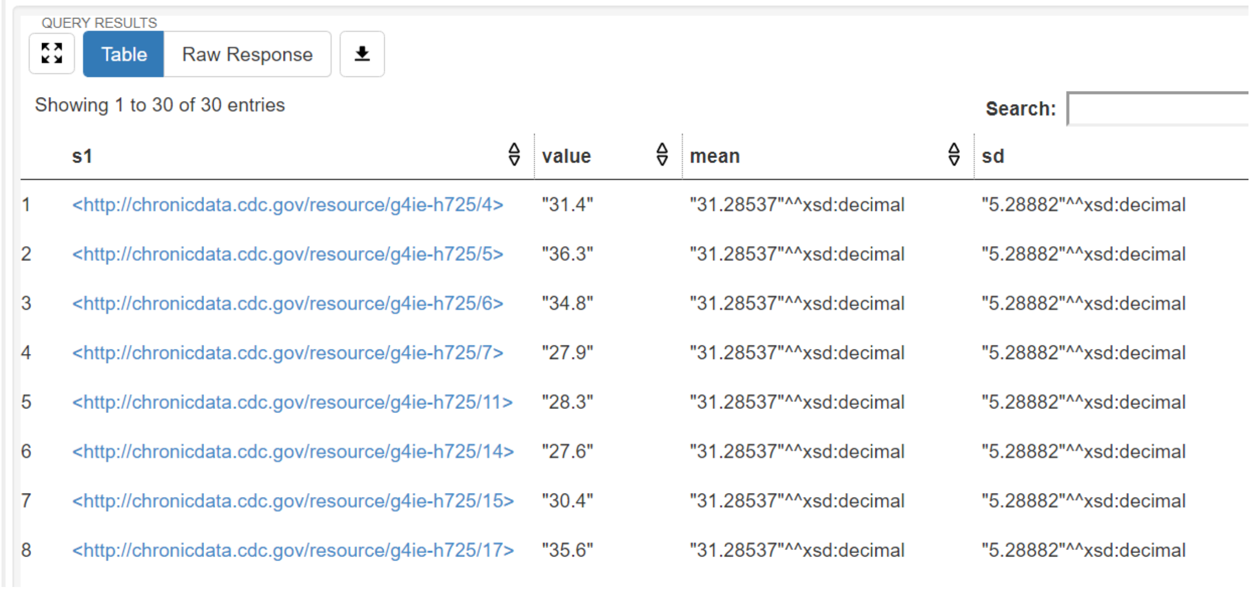


```
> query <- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ SELECT (fn:sd(?o) AS ?standardDeviation)
+ where{
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+ filter(regex(?o,'[0-9.]')) .
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
  standardDeviation
1           5.32177
```

Figure 15: SPARQL query result in R for standard deviation.

Query 7: Get the list of subjects that are within one standard deviation for "Alcohol usage among youth".

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s1 ?value ?mean ?sd
WHERE{
 ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
 ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?value .
        {
            SELECT (fn:sd(?o) AS ?sd) (fn:mean(?o) AS ?mean)
            WHERE{
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
             FILTER(regex(?o,'[0-9.]')) .
            }
  }
  FILTER(regex(?value,'[0-9.]')) .
  FILTER (xsd:decimal(?value)  <= (?mean + ?sd)) .
 FILTER (xsd:decimal(?value) >= (?mean - ?sd))
}
```

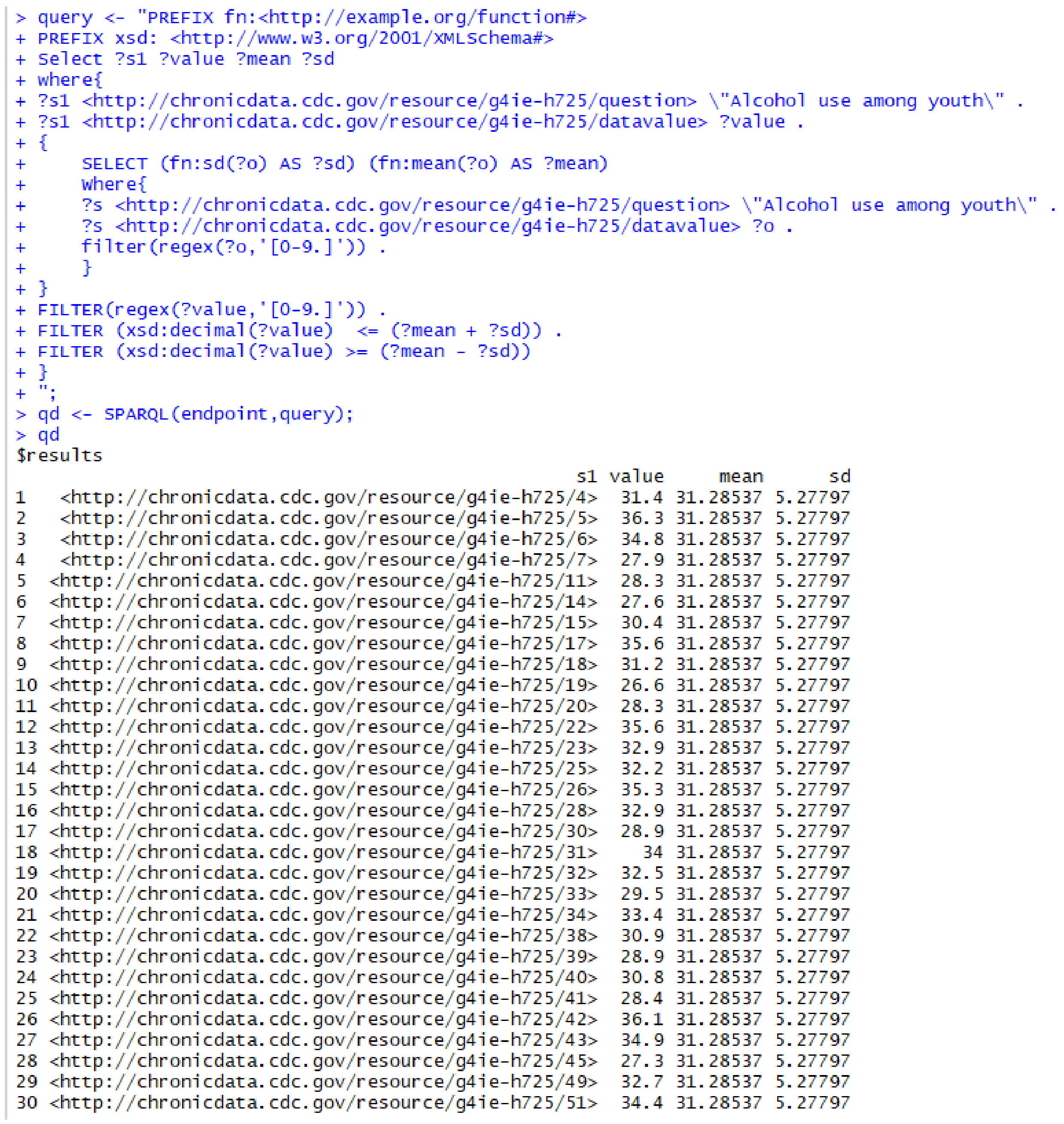


Showing 1 to 30 of 30 entries

	s1	value	mean	sd
1	<http://chronicdata.cdc.gov/resource/g4ie-h725/4>	"31.4"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
2	<http://chronicdata.cdc.gov/resource/g4ie-h725/5>	"36.3"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
3	<http://chronicdata.cdc.gov/resource/g4ie-h725/6>	"34.8"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
4	<http://chronicdata.cdc.gov/resource/g4ie-h725/7>	"27.9"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
5	<http://chronicdata.cdc.gov/resource/g4ie-h725/11>	"28.3"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
6	<http://chronicdata.cdc.gov/resource/g4ie-h725/14>	"27.6"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
7	<http://chronicdata.cdc.gov/resource/g4ie-h725/15>	"30.4"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal
8	<http://chronicdata.cdc.gov/resource/g4ie-h725/17>	"35.6"	"31.28537"^^xsd:decimal	"5.28882"^^xsd:decimal

Figure 16: Query result in Apache Jena Fuseki for subjects with alcohol usage value within one standard deviation. The result set has 30 such subjects.

When the same experiment was repeated in R, it gave similar results as shown below.

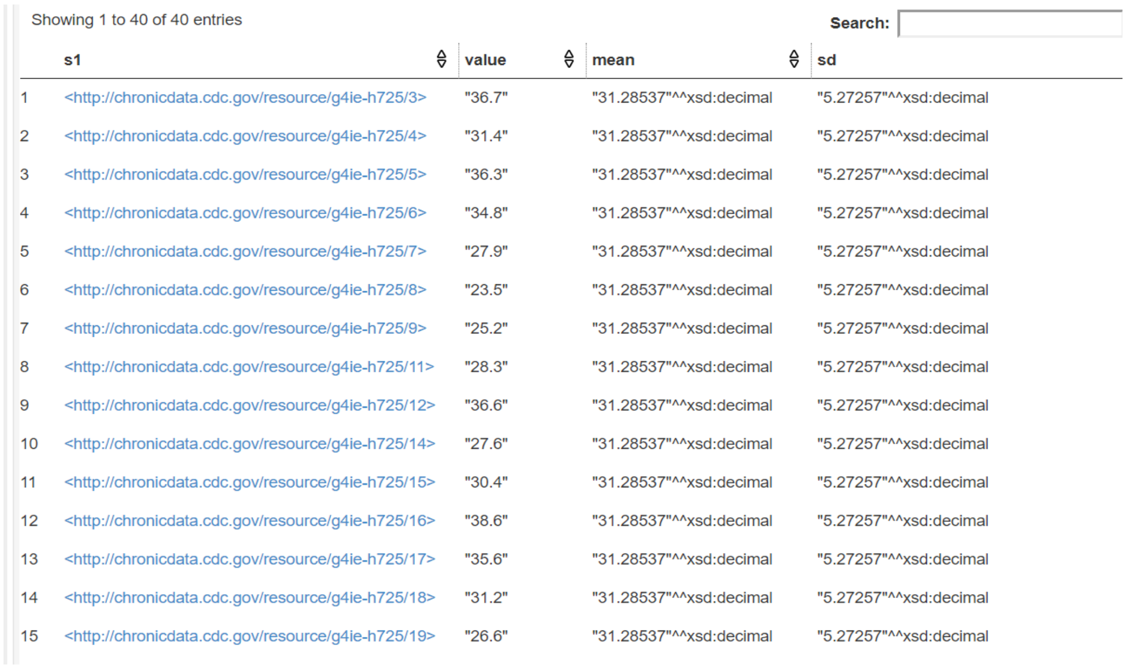


```
> query <- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ Select ?s1 ?value ?mean ?sd
+ where{
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?value .
+ {
+     SELECT (fn:sd(?o) AS ?sd) (fn:mean(?o) AS ?mean)
+     where{
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+     filter(regex(?o,'[0-9.]')) .
+     }
+ }
+ FILTER(regex(?value,'[0-9.]')) .
+ FILTER (xsd:decimal(?value)  <= (?mean + ?sd)) .
+ FILTER (xsd:decimal(?value) >= (?mean - ?sd))
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
                                                s1 value     mean      sd
1   <http://chronicdata.cdc.gov/resource/g4ie-h725/4>  31.4 31.28537 5.27797
2   <http://chronicdata.cdc.gov/resource/g4ie-h725/5>  36.3 31.28537 5.27797
3   <http://chronicdata.cdc.gov/resource/g4ie-h725/6>  34.8 31.28537 5.27797
4   <http://chronicdata.cdc.gov/resource/g4ie-h725/7>  27.9 31.28537 5.27797
5  <http://chronicdata.cdc.gov/resource/g4ie-h725/11>  28.3 31.28537 5.27797
6  <http://chronicdata.cdc.gov/resource/g4ie-h725/14>  27.6 31.28537 5.27797
7  <http://chronicdata.cdc.gov/resource/g4ie-h725/15>  30.4 31.28537 5.27797
8  <http://chronicdata.cdc.gov/resource/g4ie-h725/17>  35.6 31.28537 5.27797
9  <http://chronicdata.cdc.gov/resource/g4ie-h725/18>  31.2 31.28537 5.27797
10 <http://chronicdata.cdc.gov/resource/g4ie-h725/19>  26.6 31.28537 5.27797
11 <http://chronicdata.cdc.gov/resource/g4ie-h725/20>  28.3 31.28537 5.27797
12 <http://chronicdata.cdc.gov/resource/g4ie-h725/22>  35.6 31.28537 5.27797
13 <http://chronicdata.cdc.gov/resource/g4ie-h725/23>  32.9 31.28537 5.27797
14 <http://chronicdata.cdc.gov/resource/g4ie-h725/25>  32.2 31.28537 5.27797
15 <http://chronicdata.cdc.gov/resource/g4ie-h725/26>  35.3 31.28537 5.27797
16 <http://chronicdata.cdc.gov/resource/g4ie-h725/28>  32.9 31.28537 5.27797
17 <http://chronicdata.cdc.gov/resource/g4ie-h725/30>  28.9 31.28537 5.27797
18 <http://chronicdata.cdc.gov/resource/g4ie-h725/31>    34 31.28537 5.27797
19 <http://chronicdata.cdc.gov/resource/g4ie-h725/32>  32.5 31.28537 5.27797
20 <http://chronicdata.cdc.gov/resource/g4ie-h725/33>  29.5 31.28537 5.27797
21 <http://chronicdata.cdc.gov/resource/g4ie-h725/34>  33.4 31.28537 5.27797
22 <http://chronicdata.cdc.gov/resource/g4ie-h725/38>  30.9 31.28537 5.27797
23 <http://chronicdata.cdc.gov/resource/g4ie-h725/39>  28.9 31.28537 5.27797
24 <http://chronicdata.cdc.gov/resource/g4ie-h725/40>  30.8 31.28537 5.27797
25 <http://chronicdata.cdc.gov/resource/g4ie-h725/41>  28.4 31.28537 5.27797
26 <http://chronicdata.cdc.gov/resource/g4ie-h725/42>  36.1 31.28537 5.27797
27 <http://chronicdata.cdc.gov/resource/g4ie-h725/43>  34.9 31.28537 5.27797
28 <http://chronicdata.cdc.gov/resource/g4ie-h725/45>  27.3 31.28537 5.27797
29 <http://chronicdata.cdc.gov/resource/g4ie-h725/49>  32.7 31.28537 5.27797
30 <http://chronicdata.cdc.gov/resource/g4ie-h725/51>  34.4 31.28537 5.27797
```

Figure 17: SPARQL query executed in R for subjects with alcohol usage value within one standard deviation. The result in R is the same as in Apache Jena Fuseki. The result set has 30 observations.

Query 8: Get the list of subjects that are within two standard deviations for “Alcohol usage among youth”.

```
PREFIX fn:<http://example.org/function#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?s1 ?of ?mean ?sd
WHERE{
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use among
youth" .
  ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .

            {
            SELECT (fn:sd(?o) AS ?sd) (fn:mean(?o) AS ?mean)
            WHERE{
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
            ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
             FILTER(regex(?o,'[0-9.]')) .
            }
  }

  FILTER(regex(?of,'[0-9.]')) .
  FILTER (xsd:decimal(?of)  <= (?mean + (2*?sd))) .
  FILTER (xsd:decimal(?of) >= (?mean - (2*?sd)))
}
```


Showing 1 to 40 of 40 entries

Search:

	s1	value	mean	sd
1	<http://chronicdata.cdc.gov/resource/g4ie-h725/3>	"36.7"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
2	<http://chronicdata.cdc.gov/resource/g4ie-h725/4>	"31.4"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
3	<http://chronicdata.cdc.gov/resource/g4ie-h725/5>	"36.3"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
4	<http://chronicdata.cdc.gov/resource/g4ie-h725/6>	"34.8"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
5	<http://chronicdata.cdc.gov/resource/g4ie-h725/7>	"27.9"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
6	<http://chronicdata.cdc.gov/resource/g4ie-h725/8>	"23.5"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
7	<http://chronicdata.cdc.gov/resource/g4ie-h725/9>	"25.2"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
8	<http://chronicdata.cdc.gov/resource/g4ie-h725/11>	"28.3"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
9	<http://chronicdata.cdc.gov/resource/g4ie-h725/12>	"36.6"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
10	<http://chronicdata.cdc.gov/resource/g4ie-h725/14>	"27.6"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
11	<http://chronicdata.cdc.gov/resource/g4ie-h725/15>	"30.4"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
12	<http://chronicdata.cdc.gov/resource/g4ie-h725/16>	"38.6"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
13	<http://chronicdata.cdc.gov/resource/g4ie-h725/17>	"35.6"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
14	<http://chronicdata.cdc.gov/resource/g4ie-h725/18>	"31.2"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal
15	<http://chronicdata.cdc.gov/resource/g4ie-h725/19>	"26.6"	"31.28537"^^xsd:decimal	"5.27257"^^xsd:decimal

Figure 18: Query result in Apache Jena Fuseki for subjects with alcohol usage value within two standard deviations. The result set has 40 such subjects.

The results from R also return 40 rows as indicated in the above image.

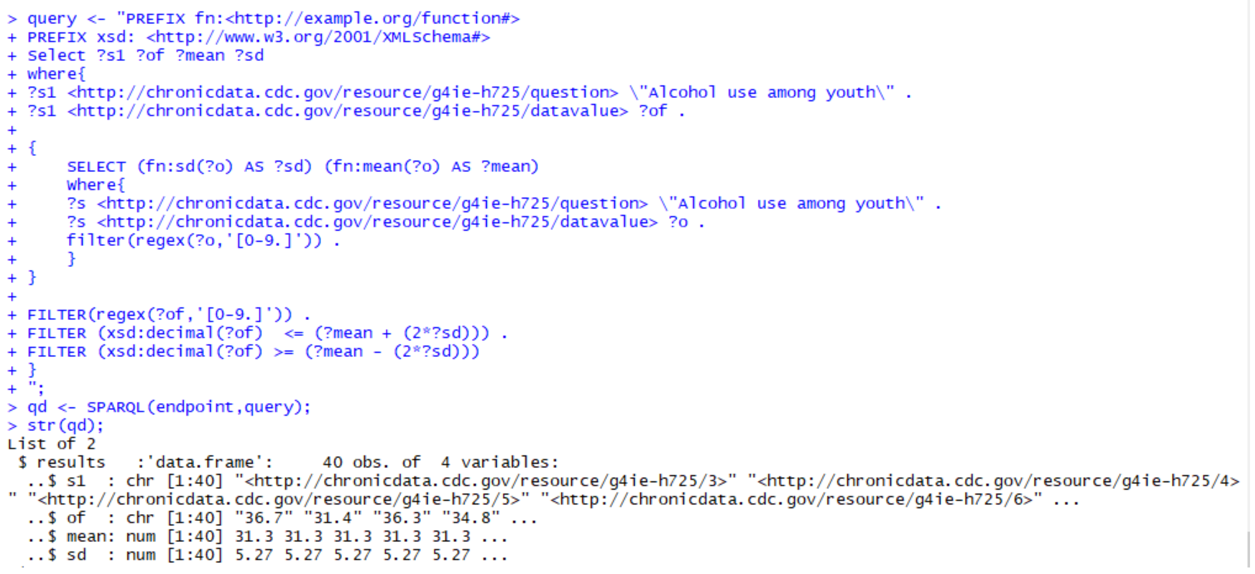

```
> query <- "PREFIX fn:<http://example.org/function#>
+ PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
+ Select ?s1 ?of ?mean ?sd
+ where{
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s1 <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?of .
+
+ {
+     SELECT (fn:sd(?o) AS ?sd) (fn:mean(?o) AS ?mean)
+     where{
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+     ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+     filter(regex(?o,'[0-9.]')) .
+     }
+ }
+
+ FILTER(regex(?of,'[0-9.]')) .
+ FILTER (xsd:decimal(?of)  <= (?mean + (2*?sd))) .
+ FILTER (xsd:decimal(?of) >= (?mean - (2*?sd)))
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> str(qd);
List of 2
 $ results   :'data.frame':	40 obs. of  4 variables:
  ..$ s1  : chr [1:40] "<http://chronicdata.cdc.gov/resource/g4ie-h725/3>" "<http://chronicdata.cdc.gov/resource/g4ie-h725/4>
" "<http://chronicdata.cdc.gov/resource/g4ie-h725/5>" "<http://chronicdata.cdc.gov/resource/g4ie-h725/6>" ...
  ..$ of  : chr [1:40] "36.7" "31.4" "36.3" "34.8" ...
  ..$ mean: num [1:40] 31.3 31.3 31.3 31.3 31.3 ...
  ..$ sd  : num [1:40] 5.27 5.27 5.27 5.27 5.27 ...
```

Figure 19: SPARQL query executed in R for subjects with alcohol usage value within two standard deviations.

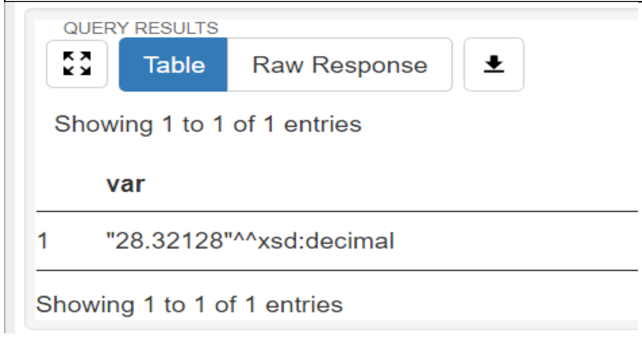
The result in R is the same as in Apache Jena Fuseki. The result set has 40 observations.

6.4 Variance

For testing the variance, the same Chronic Diseases dataset is used.

Query 9: To get the variance of the data values associated with the "Alcohol usage among youth".

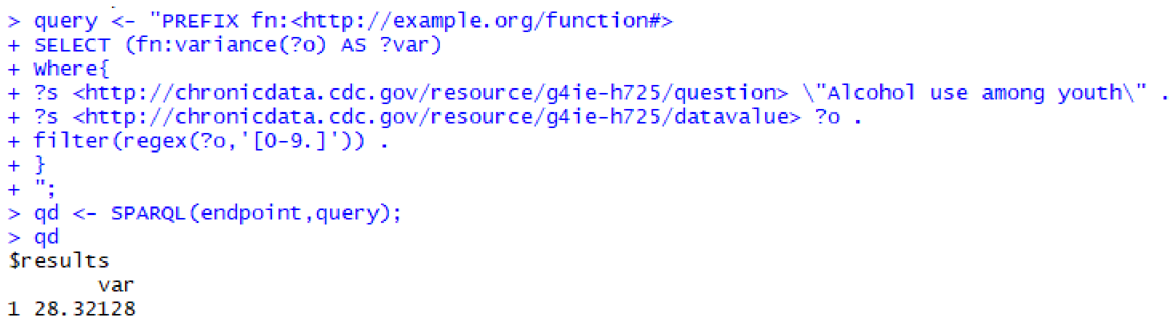
```
PREFIX fn:<http://example.org/function#>
SELECT (fn:variance(?o) AS ?var)
        WHERE{
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
         FILTER(regex(?o,'[0-9.]')) .
        }
```



```
Showing 1 to 1 of 1 entries

     var
1    "28.32128"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 20: Query result in Apache Jena Fuseki for variance.

The resultant variance when tested in R is found to be 28.32128 as shown in the figure below.



```
> query <- "PREFIX fn:<http://example.org/function#>
+ SELECT (fn:variance(?o) AS ?var)
+ where{
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+ filter(regex(?o,'[0-9.]')) .
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
       var
1 28.32128
```

Figure 21: SPARQL query executed in R for variance.

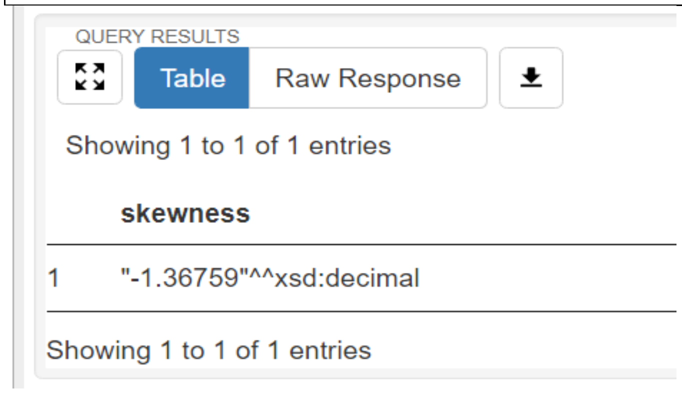
The result in R is the same as in Apache Jena Fuseki.

6.5 Skewness

Skewness characterizes the variability and measures the symmetry in the dataset. Data with normal distribution has skewness of 0. This implies that there exists a symmetry in the distribution of data. However, in reality, data that has perfect normal distribution is very rare. Below are some scenarios that show skewness in the data and to demonstrate these, we used the Chronic Diseases dataset.

Query 10: To understand how the data values related to “Alcohol usage among youth” are distributed.

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:skewness(?o) AS ?skewness)
        WHERE{
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Alcohol use
among youth" .
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
         FILTER(regex(?o,'[0-9.]')) .
        }
```



```
Showing 1 to 1 of 1 entries

     skewness
1    "-1.36759"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 22: Query result in Apache Jena Fuseki to show skewness towards the left.

A negative value of skewness indicates that the data is skewed towards the left. This implies that the tail towards the left end is longer. A histogram plot of this data indicates the same.

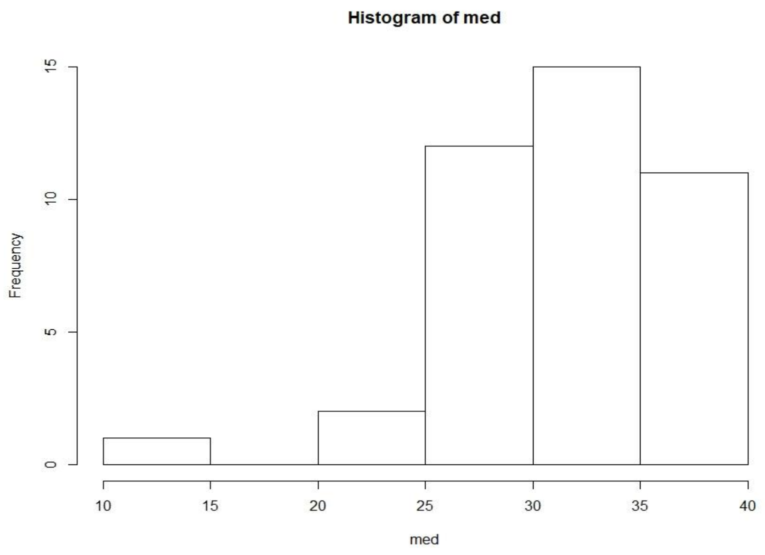
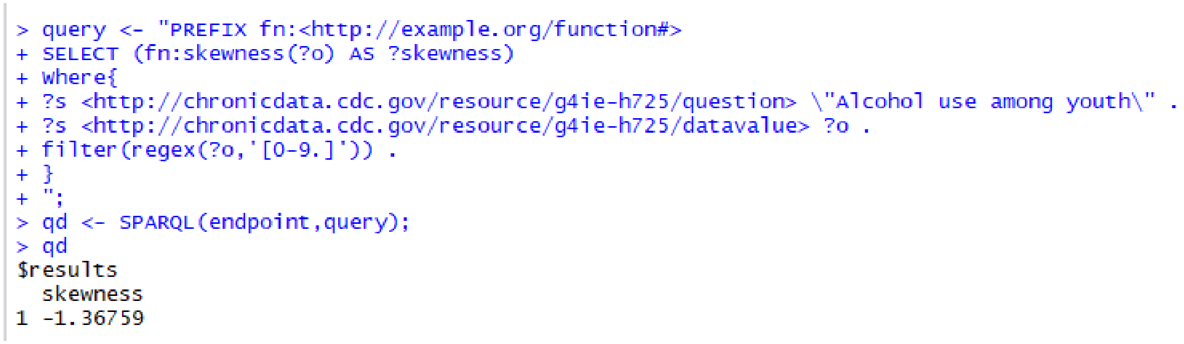


Figure 23: Histogram plot of the data values for Alcohol usage among the youth in Chronic Diseases dataset.

We repeat the same test in R and the results indicate that we got the same value in R as well.



```
> query <- "PREFIX fn:<http://example.org/function#>
+ SELECT (fn:skewness(?o) AS ?skewness)
+ where{
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Alcohol use among youth\" .
+ ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+ filter(regex(?o,'[0-9.]')) .
+ }
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
  skewness
1 -1.36759
```

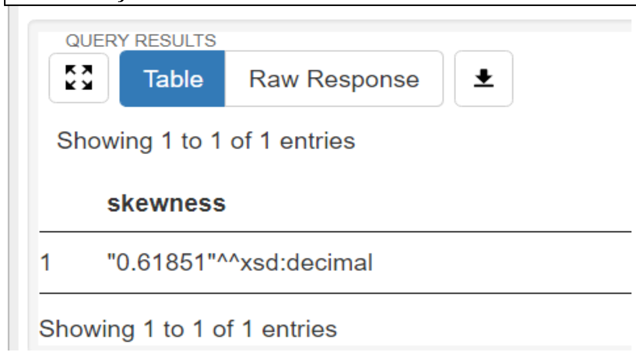
Figure 24: SPARQL query executed in R that indicates left skewed data.

The result in R is the same as in Apache Jena Fuseki.

Query 11: To get the skewness of distribution among the data values related to “Binge drinking habits among adults”.

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:skewness(?o) AS ?skewness)
        WHERE{
```

```
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> "Binge drinking
prevalence among adults aged >= 18 years" .
        ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
         FILTER(regex(?o,'[0-9.]')) .
        }
```



Showing 1 to 1 of 1 entries

	skewness
1	"0.61851"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 25: Query result in Apache Jena Fuseki with positive skewness.

The positive value here indicates that the data is slightly skewed towards the right. Since the value is less, this is nearly normally distributed data. A histogram plot in R shows the skewness in the data.

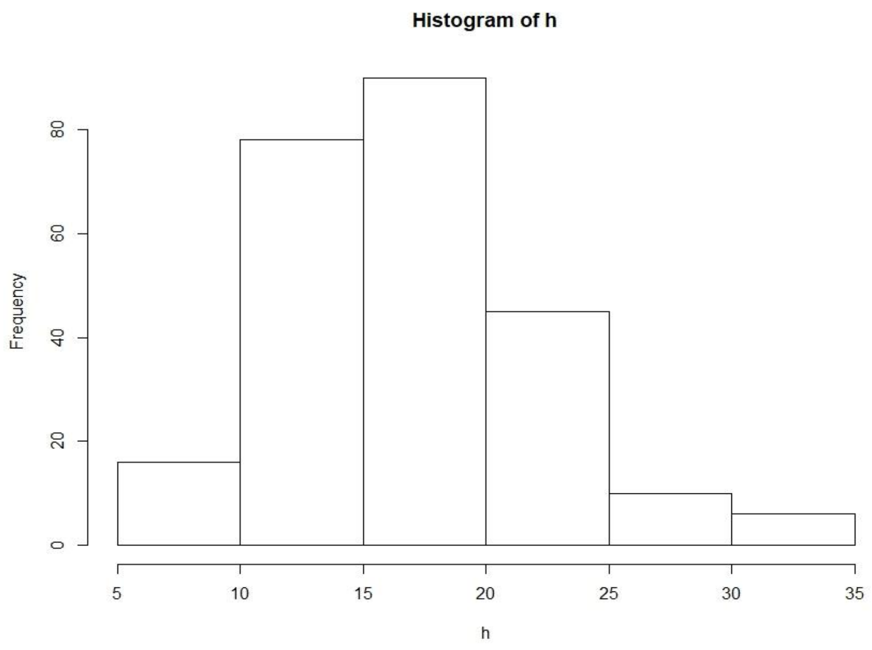
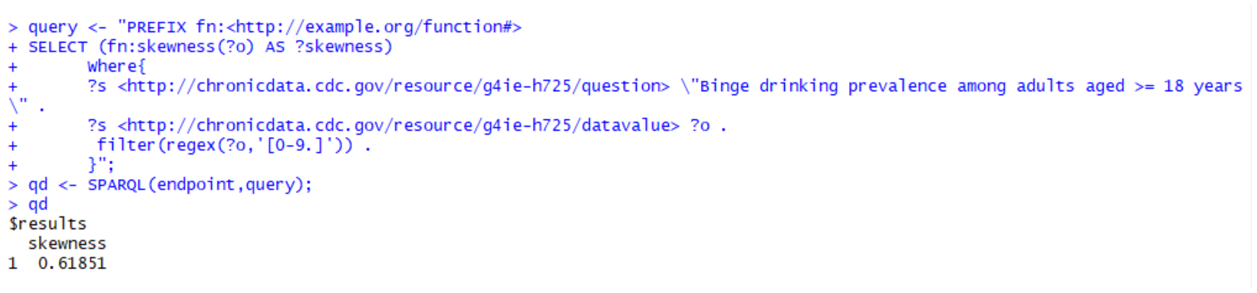


Figure 26: Histogram plot of the data values for Binge Drinking Prevalence among adults, which has a slightly longer right tail.

We repeated this test in R and results indicated that we got the same value in R as well.



```
> query <- "PREFIX fn:<http://example.org/function#>
+ SELECT (fn:skewness(?o) AS ?skewness)
+       where{
+       ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/question> \"Binge drinking prevalence among adults aged >= 18 years
\" .
+       ?s <http://chronicdata.cdc.gov/resource/g4ie-h725/datavalue> ?o .
+        filter(regex(?o,'[0-9.]')) .
+       }";
> qd <- SPARQL(endpoint,query);
> qd
$results
  skewness
1  0.61851
```

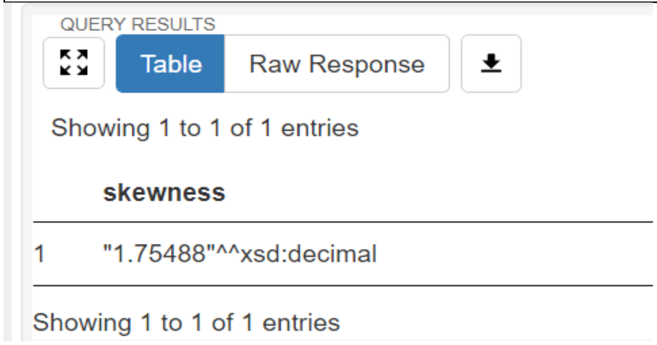
Figure 27: SPARQL query executed in R that indicates right skewed data.

The result in R is the same as in Apache Jena Fuseki.

Query 12: To get skewness in high confidence limit values in Chronic Diseases dataset.

The result value shows a clear skewness in the dataset.

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:skewness(?o) AS ?skewness)
WHERE
{
?s <http://chronicdata.cdc.gov/resource/g4ie-h725/highconfidencelimit> ?o
}
```



Showing 1 to 1 of 1 entries

	skewness
1	"1.75488"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 28: Skewness query result in Apache Jena Fuseki to indicate longer right tail.

A histogram plot in R shows that the right tail is longer and the data is skewed towards the right.

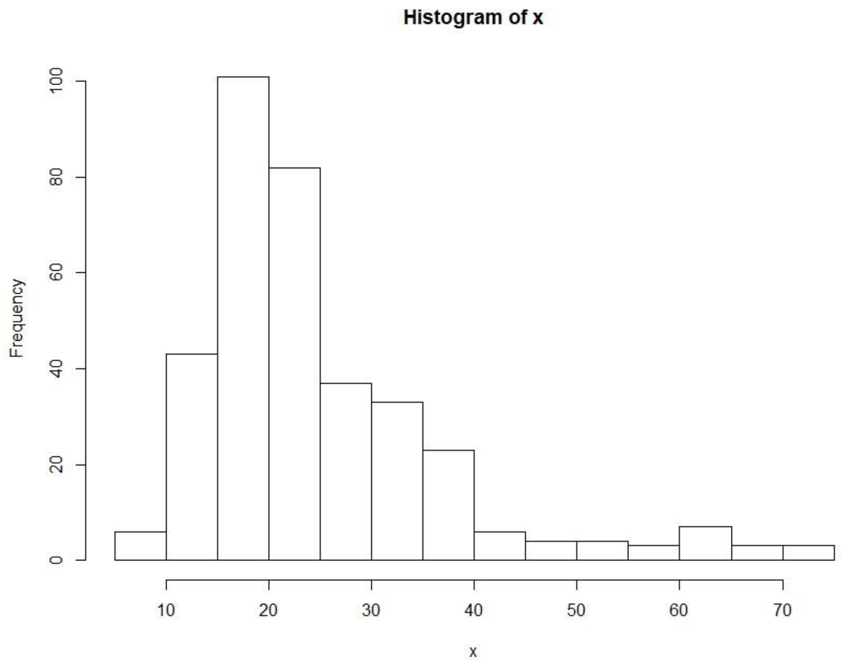


Figure 29: Histogram plot of the data values used for high confidence limit to show a longer right tail.

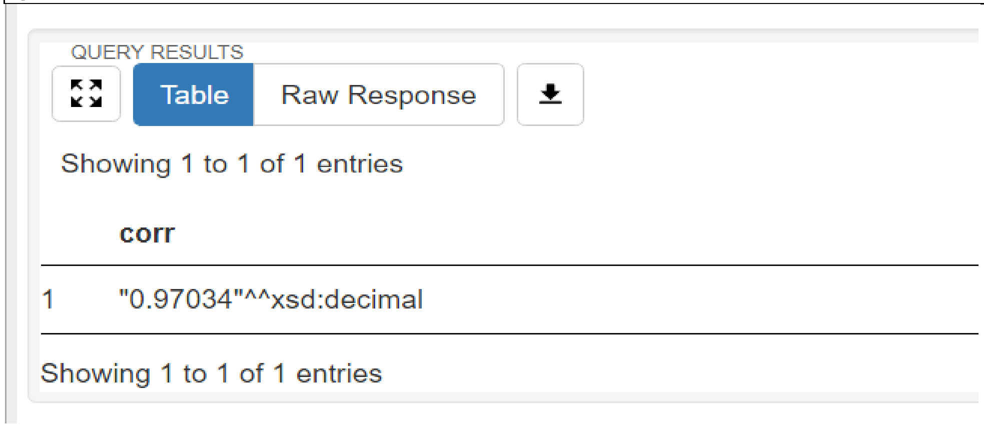
6.6 Pearson Correlation

As part of evaluating correlation, we are testing Pearson Correlation, Spearman Correlation and Kendall Correlation. Some of the datasets are taken from Data.gov and some are constructed as RDF documents. We have constructed the RDF documents explicitly so that the dataset has readings for the same subjects. We consider two variables for correlation testing wherein one is treated as an independent variable and the other is treated as a dependent variable. We measure both strength and direction of the correlation.

In this section we test the Pearson Correlation. The dataset we are considering here is related to SAT test results for the year 2012 in New York city (https://catalog.data.gov/dataset/sat-results-e88d7). We are comparing the correlations between reading and writing scores.

Query 13: Pearson correlation to understand the correlation between SAT critical reading and writing scores.

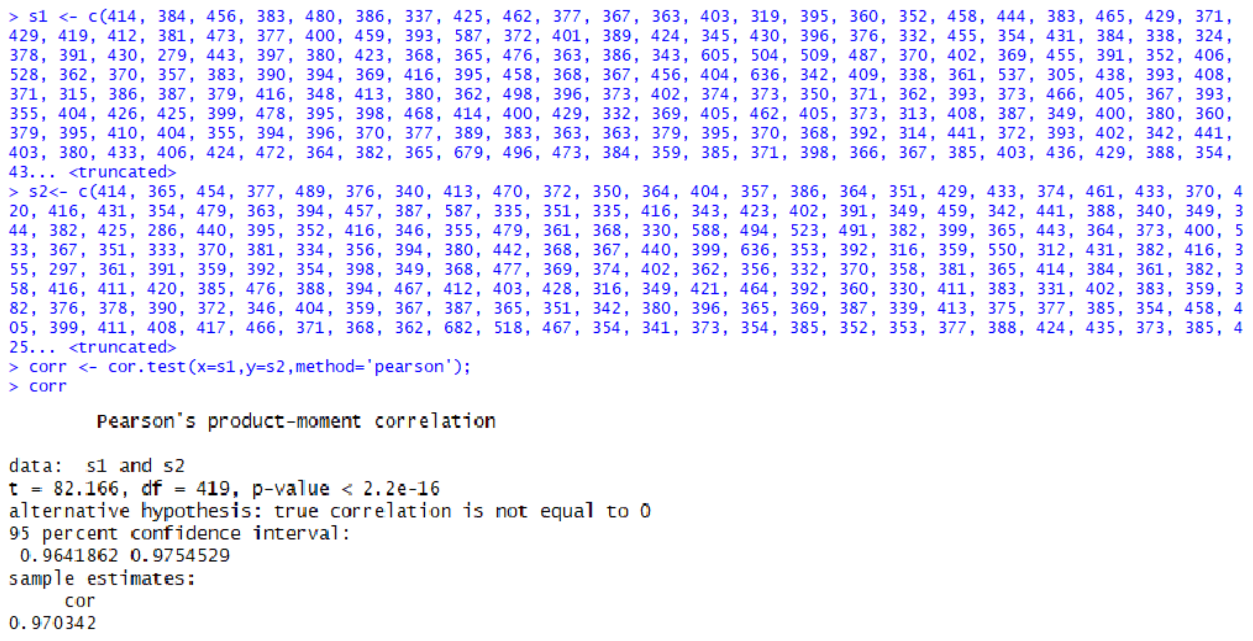
```
PREFIX fn:<http://example.org/function#>
SELECT
(fn:pearsoncorr(group_concat(?o1;separator=':'),group_concat(?o2;separator=':')) AS
?corr)
WHERE{
?s <https://data.cityofnewyork.us/resource/f9bf-2cp4/sat_critical_reading_avg_score>
?o1 .
?s <https://data.cityofnewyork.us/resource/f9bf-2cp4/sat_writing_avg_score> ?o2 .
 FILTER(regex(?o1,'[0-9.]'))  .
 FILTER(regex(?o2,'[0-9.]'))  .
}
```



```
Showing 1 to 1 of 1 entries

     corr
1    "0.97034"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 30: Pearson correlation query result in Apache Jena Fuseki for SAT dataset.

This value indicates a very positive correlation between SAT reading and writing scores as the result is close to 1. The same test is performed in R by taking the data-values of critical reading and writing scores as two vectors.



```
> s1 <- c(414, 384, 456, 383, 480, 386, 337, 425, 462, 377, 367, 363, 403, 319, 395, 360, 352, 458, 444, 383, 465, 429, 371,
429, 419, 412, 381, 473, 377, 400, 459, 393, 587, 372, 401, 389, 424, 345, 430, 396, 376, 332, 455, 354, 431, 384, 338, 324,
378, 391, 430, 279, 443, 397, 380, 423, 368, 365, 476, 363, 386, 343, 605, 504, 509, 487, 370, 402, 369, 455, 391, 352, 406,
528, 362, 370, 357, 383, 390, 394, 369, 416, 395, 458, 368, 367, 456, 404, 636, 342, 409, 338, 361, 537, 305, 438, 393, 408,
371, 315, 386, 387, 379, 416, 348, 413, 380, 362, 498, 396, 373, 402, 374, 373, 350, 371, 362, 393, 373, 466, 405, 367, 393,
355, 404, 426, 425, 399, 478, 395, 398, 468, 414, 400, 429, 332, 369, 405, 462, 405, 373, 313, 408, 387, 349, 400, 380, 360,
379, 395, 410, 404, 355, 394, 396, 370, 377, 389, 383, 363, 363, 379, 395, 370, 368, 392, 314, 441, 372, 393, 402, 342, 441,
403, 380, 433, 406, 424, 472, 364, 382, 365, 679, 496, 473, 384, 359, 385, 371, 398, 366, 367, 385, 403, 436, 429, 388, 354,
43... <truncated>
> s2<- c(414, 365, 454, 377, 489, 376, 340, 413, 470, 372, 350, 364, 404, 357, 386, 364, 351, 429, 433, 374, 461, 433, 370, 4
20, 416, 431, 354, 479, 363, 394, 457, 387, 587, 335, 351, 335, 416, 343, 423, 402, 391, 349, 459, 342, 441, 388, 340, 349, 3
44, 382, 425, 286, 440, 395, 352, 416, 346, 355, 479, 361, 368, 330, 588, 494, 523, 491, 382, 399, 365, 443, 364, 373, 400, 5
33, 367, 351, 333, 370, 381, 334, 356, 394, 380, 442, 368, 367, 440, 399, 636, 353, 392, 316, 359, 550, 312, 431, 382, 416, 3
55, 297, 361, 391, 359, 392, 354, 398, 349, 368, 477, 369, 374, 402, 362, 356, 332, 370, 358, 381, 365, 414, 384, 361, 382, 3
58, 416, 411, 420, 385, 476, 388, 394, 467, 412, 403, 428, 316, 349, 421, 464, 392, 360, 330, 411, 383, 331, 402, 383, 359, 3
82, 376, 378, 390, 372, 346, 404, 359, 367, 387, 365, 351, 342, 380, 396, 365, 369, 387, 339, 413, 375, 377, 385, 354, 458, 4
05, 399, 411, 408, 417, 466, 371, 368, 362, 682, 518, 467, 354, 341, 373, 354, 385, 352, 353, 377, 388, 424, 435, 373, 385, 4
25... <truncated>
> corr <- cor.test(x=s1,y=s2,method='pearson');
> corr

        Pearson's product-moment correlation

data:  s1 and s2
t = 82.166, df = 419, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9641862 0.9754529
sample estimates:
      cor
0.970342
```

Figure 31: Pearson correlation calculated in R for SAT dataset.

The data values are stored in vectors and we utilized R's inbuilt functions to calculate the Pearson coefficient.

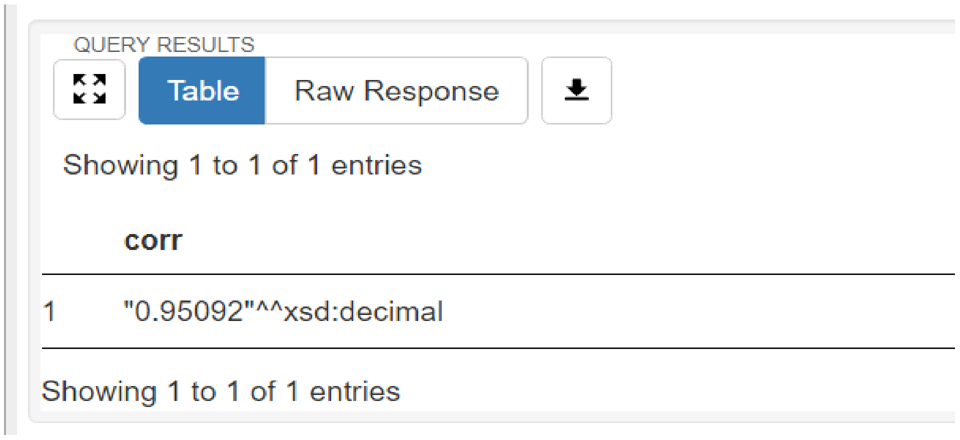
The second dataset we are considering is cars dataset. This dataset is taken from http://www-bcf.usc.edu/~gareth/ISL/data.html. This dataset is available in csv format, but we converted it to RDF format for our analysis.

Query 14: Pearson correlation between the number of cylinders and the displacement in a car.

We are trying to understand the relationship between the number of cylinders in a car and the displacement the car offers. Here, the number of cylinders is an independent variable for the Pearson correlation and the displacement is a dependent variable for this correlation.

```
PREFIX fn:<http://example.org/function#>
SELECT
(fn:pearsoncorr(group_concat(?cy;separator=':'),group_concat(?di;separator=':')) AS
?corr)
            WHERE
            {
```

```
            ?s <http://mdsp.org/data/Auto#cylinders> ?cy .
            ?s <http://mdsp.org/data/Auto#displacement> ?di .
            FILTER(regex(?di ,'[0-9.]')) .
            FILTER(regex(?cy ,'[0-9.]'))
            }
```



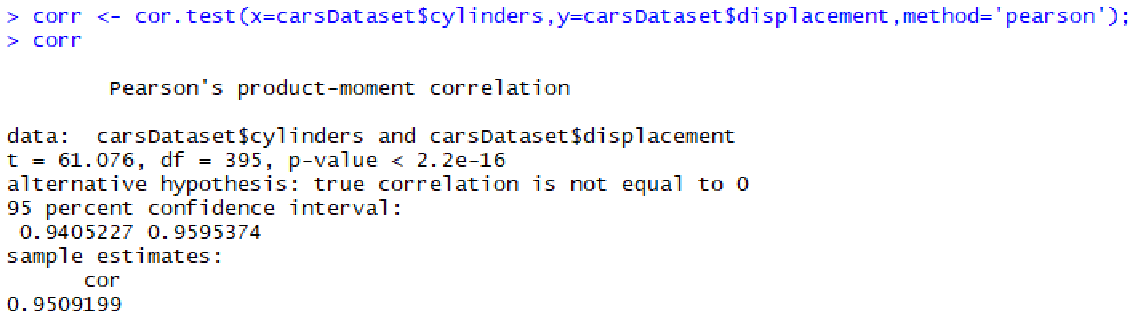
Showing 1 to 1 of 1 entries

	corr
1	"0.95092"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 32: Pearson correlation query result in Apache Jena Fuseki for Automobile dataset which indicates positive correlation.

Since the result is close to 1, we can conclude that there is a high correlation between the number of cylinders in a car and the displacement it offers. The same test is repeated in R and we obtained similar results as indicated below.



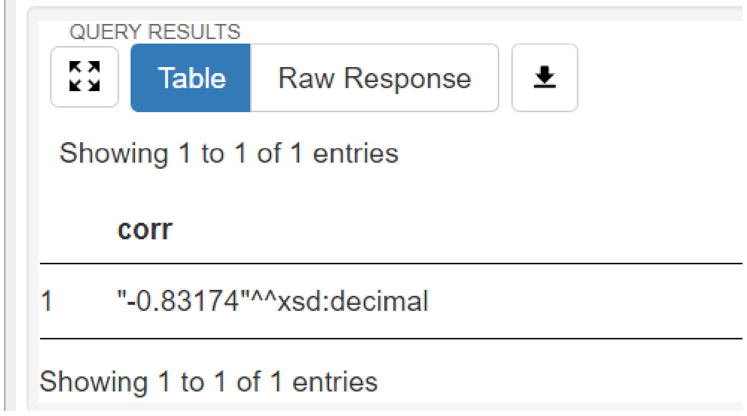
```
> corr <- cor.test(x=carsDataset$cylinders,y=carsDataset$displacement,method='pearson');
> corr

        Pearson's product-moment correlation

data:  carsDataset$cylinders and carsDataset$displacement
t = 61.076, df = 395, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9405227 0.9595374
sample estimates:
      cor
0.9509199
```

Figure 33: Pearson correlation calculated in R using R’s inbuilt function for Automobile dataset indicating positive correlation.

The dataset is loaded as a csv file in R and computations are performed on the values read from the csv file.

Query 15: Relation between the weight and the miles per gallon (mpg). Here, weight is the independent variable and the mpg is the dependent variable.

```
PREFIX fn:<http://example.org/function#>
SELECT
(fn:pearsoncorr(group_concat(?mpg;separator=':'),group_concat(?wt;separator=':')) AS
?corr)
            WHERE
            {
            ?s <http://mdsp.org/data/Auto#mpg> ?mpg .
            ?s <http://mdsp.org/data/Auto#weight> ?wt .
            FILTER(regex(?mpg ,'[0-9.]')) .
            FILTER(regex(?wt ,'[0-9.]'))
            }
```

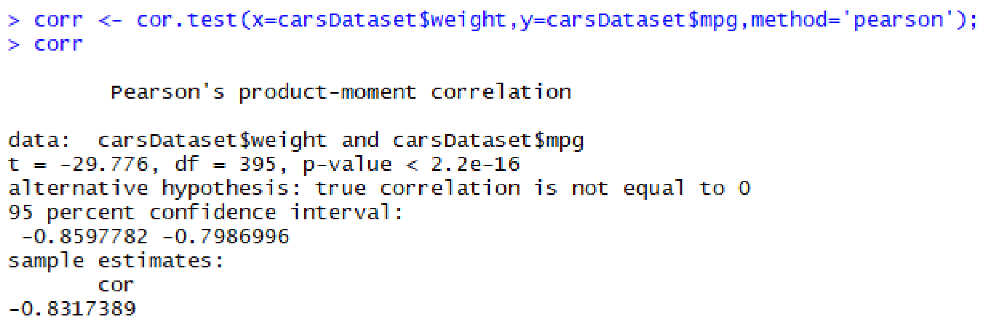


```
Showing 1 to 1 of 1 entries

     corr
1    "-0.83174"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 34: Pearson correlation query result in Apache Jena Fuseki for SAT dataset to show negative correlation.

Since the value is negative and close to -1, we conclude that there is a strong inverse relationship between the weight of the car and the mpg. Stated in other terms, as the weight of the car increases, the mpg value decreases. So, we conclude that heavier cars give less mileage.

This test is repeated in R and it gives similar results as indicated below.



```
> corr <- cor.test(x=carsDataset$weight,y=carsDataset$mpg,method='pearson');
> corr

	Pearson's product-moment correlation

data:  carsDataset$weight and carsDataset$mpg
t = -29.776, df = 395, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.8597782 -0.7986996
sample estimates:
       cor 
-0.8317389 
```

Figure 35: Pearson correlation calculated in R using R's inbuilt function for Automobile dataset indicating negative correlation.

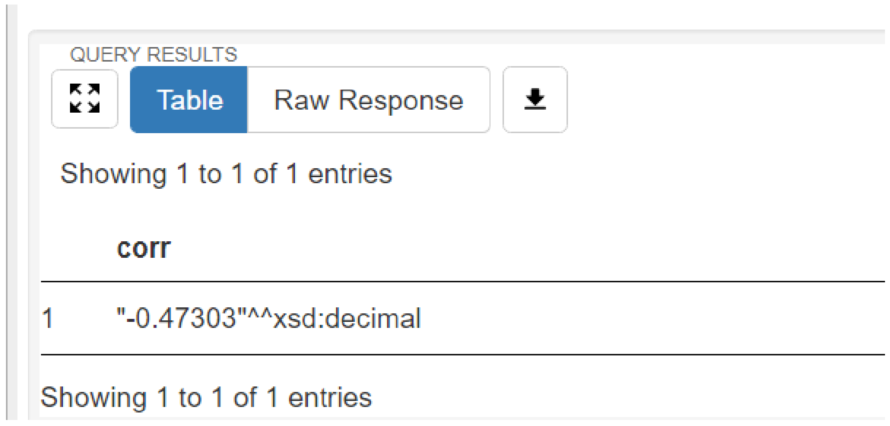
6.7 Spearman Correlation

Spearman correlation is used when one of the variables is ranked and the data values don't follow a bivariate normal distribution. Instead of testing a linear relationship, spearman correlation helps us in testing monotonic relations. For evaluating Spearman correlation, we are using datasets that have information on cars. Spearman takes one of the variables as a ranked variable.

Query 16: Relation between the number of cylinders and acceleration that a car offers.

There is a repetition of values for number of cylinders in the dataset as cars can have 4, 6 or 8 cylinders. Hence, we can consider the ranking of the cylinders variable and test the correlation using spearman correlation. The ranked variable is given as the first parameter.

```
PREFIX fn:<http://example.org/function#>
SELECT
(fn:spearmancorr(group_concat(?cy;separator=':'),group_concat(?ac;separator=':')) AS
?corr)
		WHERE
		{
		?s <http://mdsp.org/data/Auto#cylinders> ?cy .
		?s <http://mdsp.org/data/Auto#acceleration> ?ac .
		FILTER(regex(?cy ,'[0-9.]')) .
		FILTER(regex(?ac ,'[0-9.]'))
		}
```

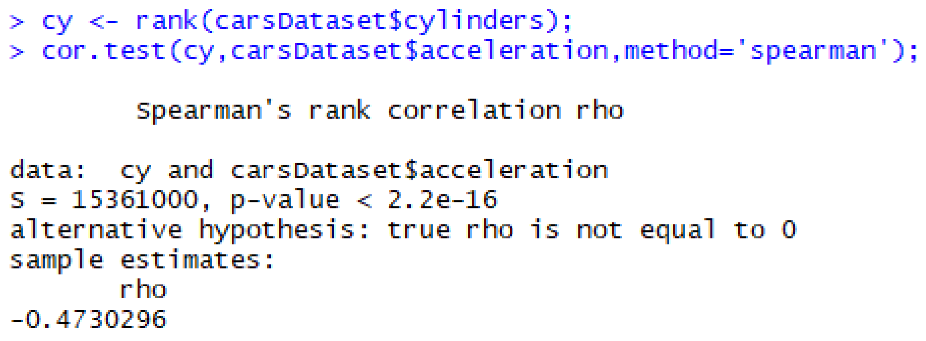


```
QUERY RESULTS
Table   Raw Response

Showing 1 to 1 of 1 entries

      corr
1     "-0.47303"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 36: Spearman correlation query result in Apache Jena Fuseki for Automobile dataset indicating insignificant negative correlation.

Since the result is negative, we can say that there is inverse relation. But the value indicates that the strength of correlation is not enough to conclude that the two variables are inversely related.

We repeat the same test in R and, as shown below, the test gives the same result.



```
> cy <- rank(carsDataset$cylinders);
> cor.test(cy,carsDataset$acceleration,method='spearman');

        Spearman's rank correlation rho

data:  cy and carsDataset$acceleration
S = 15361000, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
       rho 
-0.4730296 
```

Figure 37: Spearman correlation calculated in R using R's inbuilt function for Automobile dataset indicating insignificant negative correlation.

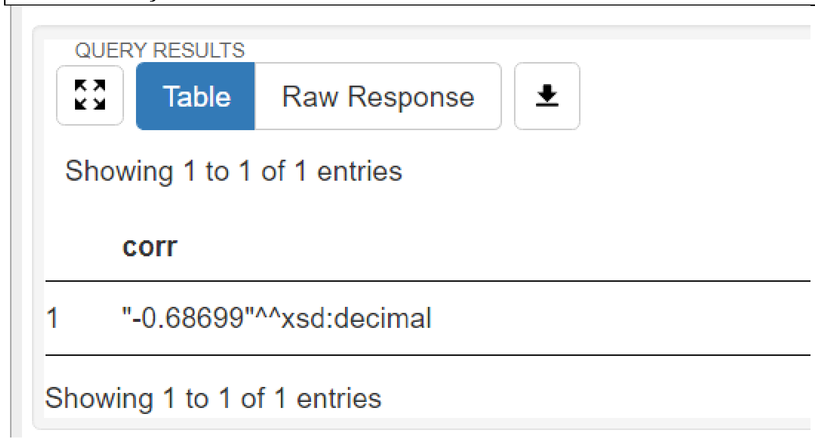
6.8 Kendall Correlation

We perform Kendall correlation between two ranked variables. This way we understand the concordant and discordant pairs. Concordant pairs are how many larger ranks are below a certain rank. Discordant pairs indicate how the variables' values differ in ranking. Here, we consider the ranking among the data values for the two variables. A perfect correlation exists when both values get same order of ranks.

For testing the Kendall correlation, we are using the same cars dataset that we used in the Pearson and the Spearman correlations.

Query 17: To find the correlation between the number of cylinders and the mpg. Here, cylinders and mpg both have repetitive data values. Hence, we can rank them in order to understand the correlation in a better way.

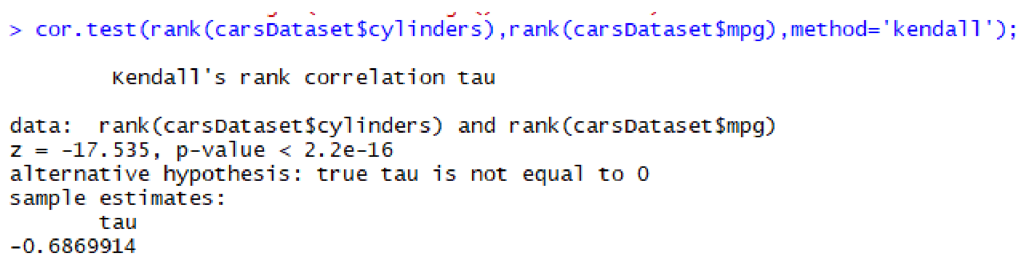
```
PREFIX fn:<http://example.org/function#>
SELECT
(fn:kendallcorr(group_concat(?cy;separator=':'),group_concat(?mpg;separator=':')) AS
?corr)
        WHERE
        {
        ?s <http://mdsp.org/data/Auto#cylinders> ?cy .
        ?s <http://mdsp.org/data/Auto#mpg> ?mpg .
        FILTER(regex(?cy ,'[0-9.]')) .
        FILTER(regex(?mpg ,'[0-9.]'))
        }
```



```
Showing 1 to 1 of 1 entries

     corr
1    "-0.68699"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 38: Kendall correlation query result in Apache Jena Fuseki for Automobile dataset indicating negative correlation.

This value suggests that there is a significant inverse relationship between the number of cylinders and the mpg. We repeat the same test in R and the results are shown below.



```
> cor.test(rank(carsDataset$cylinders),rank(carsDataset$mpg),method='kendall');

        Kendall's rank correlation tau

data:  rank(carsDataset$cylinders) and rank(carsDataset$mpg)
z = -17.535, p-value < 2.2e-16
alternative hypothesis: true tau is not equal to 0
sample estimates:
       tau
-0.6869914
```

Figure 39: Kendall correlation calculated in R using R’s inbuilt function for Automobile dataset indicating negative correlation.

6.9 Z Test

In R, there is no pre-defined function for evaluating Z Test. We are writing R scripts in Java and calling these scripts from SPARQL user-defined functions.

For conducting Z test, the dataset from Center for Disease Control is utilized.

(https://catalog.data.gov/dataset/behavioral-risk-factor-data-tobacco-use-2011-to-present-e0ad1)

Query 18: One sampled upper tailed Z test

Hypothesis: The average tobacco usage in Georgia has increased between 2014-2015. It was observed that the average tobacco usage among adult population between 2013-2014 was 17.576%. The standard deviation of the population was 4.21377. The sample size we are considering is 30.

From the above hypothesis, we state the null hypothesis and the alternative hypothesis and take a confidence value of 95%.

Null Hypothesis H_0: $\mu = 17.576$

Alternate Hypothesis H_a: $\mu > 17.576$

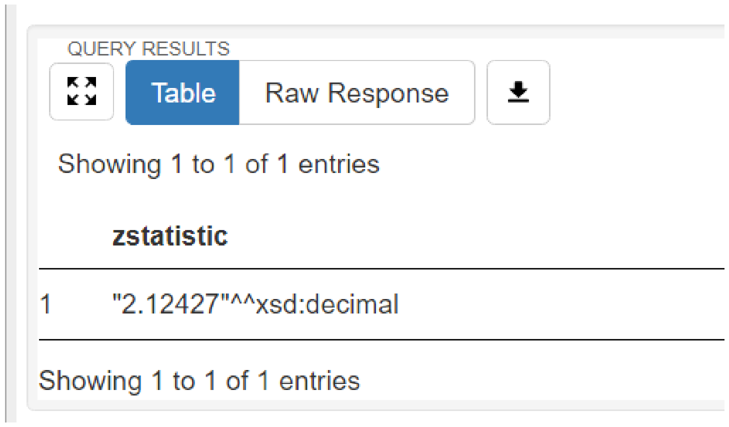
α: 0.05 (Taken from 95% confidence). Z score at 95 percent confidence = 1.64

Analysis plan: If the z score of our evaluation is greater than 1.64, our value lies in the rejection region and hence we reject the null hypothesis.

Since we are stating that the average has increased, we are conducting an upper-tailed Z test.

Query:

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:oneZTest(group_concat(?o;separator=','),17.57692,17.75583,30) AS
?zstatistic)
WHERE {
        ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/locationabbr> "GA" .
        ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/year> "2014-2015"  .
        ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/topicdesc> "Cigarette Use
(Adults)" .
        ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/data_value> ?o .
FILTER(regex(?o ,'[0-9.]'))
}
```



Showing 1 to 1 of 1 entries

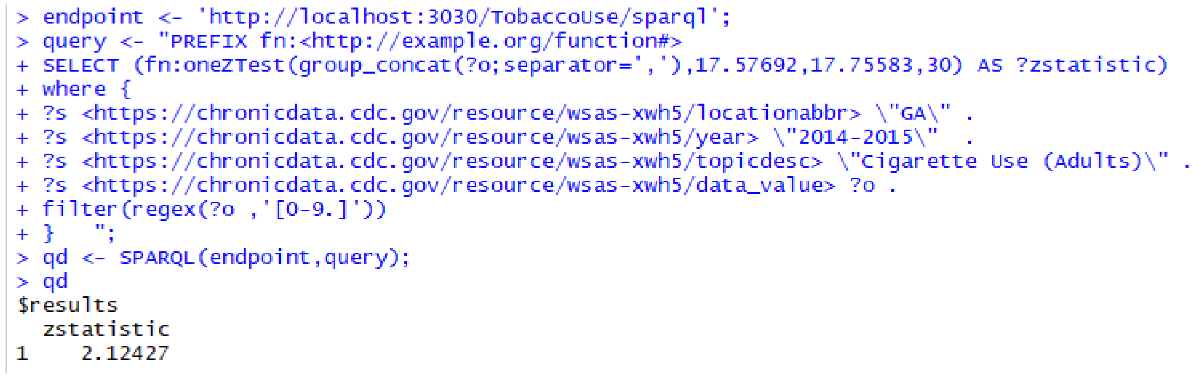
	zstatistic
1	"2.12427"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 40: One sample Z test query result in Apache Jena Fuseki for Tobacco usage dataset.

Since z score is greater than 1.64, we reject the null hypothesis and conclude that the average tobacco usage has increased.

The query when executed in R gives us the same result as shown below.



```
> endpoint <- 'http://localhost:3030/TobaccoUse/sparql';
> query <- "PREFIX fn:<http://example.org/function#>
+ SELECT (fn:oneZTest(group_concat(?o;separator=','),17.57692,17.75583,30) AS ?zstatistic)
+ where {
+ ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/locationabbr> \"GA\" .
+ ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/year> \"2014-2015\"  .
+ ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/topicdesc> \"Cigarette Use (Adults)\" .
+ ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/data_value> ?o .
+ filter(regex(?o ,'[0-9.]'))
+ }   ";
> qd <- SPARQL(endpoint,query);
> qd
$results
  zstatistic
1    2.12427
```

Figure 41: One sample Z test calculated in R for Tobacco usage dataset.

Query 19: Two samples lower tailed Z test

Hypothesis: Average tobacco consumption in Alabama is lower than the average tobacco consumption in Georgia. A sample of 35 members is drawn from both datasets. This test can be performed if we know the standard deviation or variance of both datasets. The variance of the Georgia dataset is 17.75583. The variance of the Alabama dataset is 25.082.

Null hypothesis: $\mu_1 = \mu_2$

Alternative Hypothesis: $\mu_1 < \mu_2$

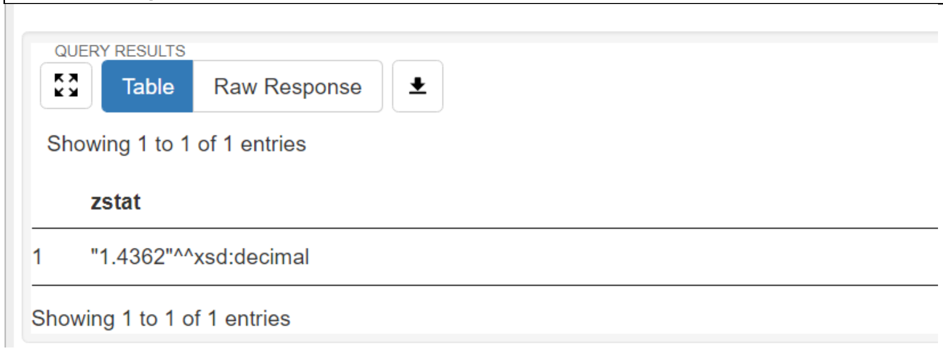
α: 0.05 (Taken from 95% confidence). Z score at 95 percent confidence = 1.64

Analysis plan: If the Z score of our evaluation is less than -1.64, our value lies in the rejection region and hence we reject the null hypothesis.

Since we are stating that the average of the first set is less than the second set, we are conducting lower-tailed Z test for two samples.

Query:

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:twoZTest(?oalConcat,?ogaConcat,25.082,17.75583,35) AS ?zstat)
WHERE{
            {
           SELECT (group_concat(?oal;separator=':') AS ?oalConcat)
                      WHERE{
                         ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/locationabbr>
"AL" .
                         ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/year> "2013-
2014"  .
                         ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/topicdesc>
"Cigarette Use (Adults)" .
                         ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/measuredesc>
"Current Smoking – (2 yrs – Race/Ethnicity)" .
                         ?s <https://chronicdata.cdc.gov/resource/wsas-xwh5/data_value>
?oal .
                         filter(regex(?oal ,'[0-9.]')) .
                         }
              }
              {
            SELECT (group_concat(?oga;separator=':') AS ?ogaConcat)
            WHERE{
                     ?s1 <https://chronicdata.cdc.gov/resource/wsas-xwh5/locationabbr>
"GA" .
                         ?s1 <https://chronicdata.cdc.gov/resource/wsas-xwh5/year> "2013-
2014"  .
                         ?s1 <https://chronicdata.cdc.gov/resource/wsas-xwh5/topicdesc>
"Cigarette Use (Adults)" .
                         ?s1 <https://chronicdata.cdc.gov/resource/wsas-
xwh5/measuredesc> "Current Smoking – (2 yrs – Race/Ethnicity)" .
                         ?s1 <https://chronicdata.cdc.gov/resource/wsas-xwh5/data_value>
?oga .
                         FILTER(regex(?oga ,'[0-9.]'))
                  }
            }
            }
```

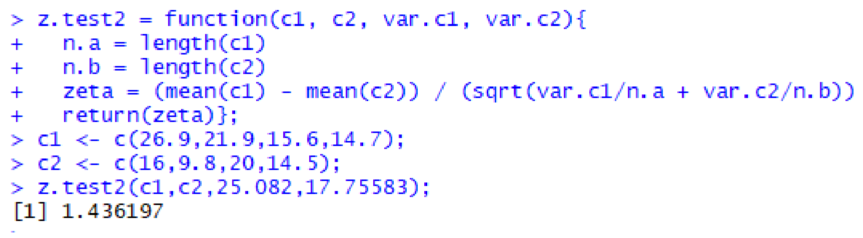


```
Showing 1 to 1 of 1 entries

     zstat
1    "1.4362"^^xsd:decimal

Showing 1 to 1 of 1 entries
```

Figure 42: Two samples Z test query result in Apache Jena Fuseki for Tobacco Usage dataset.

As per our analysis plan, we accept the null hypothesis since the z value lies in the acceptance region i.e., z > -1.64. And we conclude that the average tobacco consumption in Alabama is not lower than the average tobacco consumption in Georgia, i.e., both have similar consumption rates.

We repeated the same test in R. Instead of running a SPARQL query, we used R's z-test function to test the hypothesis as shown below. This is to check if there are any anomalies by creating the vectors with the same data values.



```
> z.test2 = function(c1, c2, var.c1, var.c2){
+   n.a = length(c1)
+   n.b = length(c2)
+   zeta = (mean(c1) - mean(c2)) / (sqrt(var.c1/n.a + var.c2/n.b))
+   return(zeta)};
> c1 <- c(26.9,21.9,15.6,14.7);
> c2 <- c(16,9.8,20,14.5);
> z.test2(c1,c2,25.082,17.75583);
[1] 1.436197
```

Figure 43: Two samples Z test calculated in R for Tobacco Usage dataset.

Here, we performed the computations local in R by writing a function for Z test since R doesn't have any inbuilt package function to calculate Z test. The result indicates that the value in R environment is equal to the value we got in the SPARQL query.

6.10 T-Test

T-test is performed when the sample size is small, and we don't know the standard deviation. The 1-sample T test compares the mean score of an observation sample to a hypothetically assumed value. The hypothetical value is the population mean where as the observed mean is noted for a sample of the population. The 2-sample T test compares the two samples to compare unrelated groups.

Query20: One sample upper tailed T test

Hypothesis: The average rate of prevalence of cardiovascular diseases among US adults between 2001-2002 is observed to be 10.10714. However, researchers believe that this has increased in the duration of 2003-2004. Our hypothesis test is to understand if the average rate of prevalence of cardiovascular diseases among US adults has increased and what is the significance. The data set we are using here is Nutrition Examination Survey data. (https://catalog.data.gov/dataset/national-health-and-nutrition-examination-survey-nhanes)

Null Hypothesis: μ= 10.10714

Alternative Hypothesis: μ > 10.10714

α: Here we assume 5%. i.e., confidence level = 95 percent.

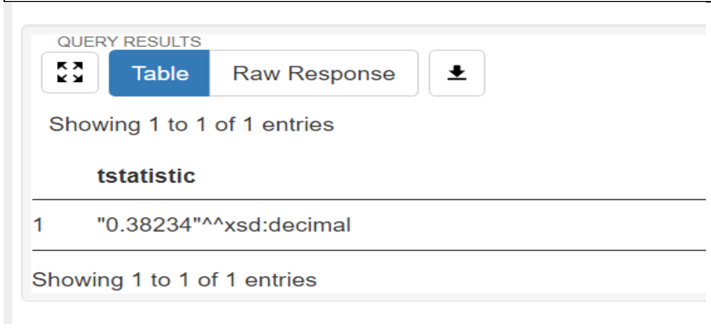
Sample size = 13

Degrees of freedom: 12 (Sample size -1)

Analysis result: If the resultant T score is greater than 1.782 (as observed from the t table), we reject Null hypothesis. Otherwise, we accept the null hypothesis.

Query

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:oneTTest(group_concat(?o;separator=','),10.10714,13) AS ?tstatistic)
WHERE{
 ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/indicator> "Prevalence of major cardiovascular disease among US adults (20+); NHANES" .
 ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/year> "2003-2004" .
 ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/data_value> ?o
}
```



Showing 1 to 1 of 1 entries

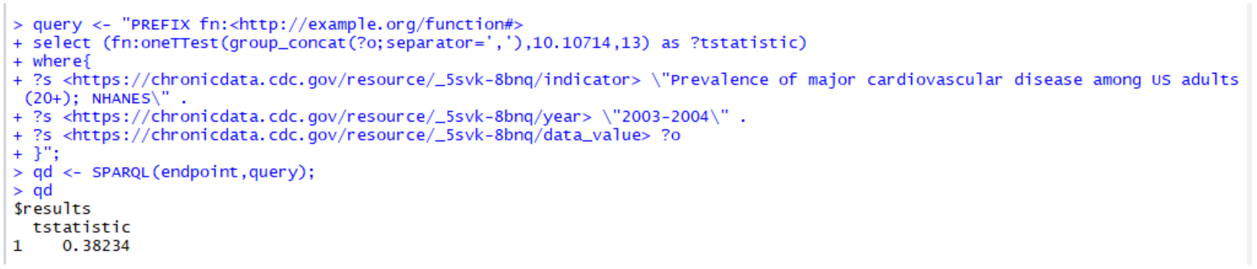
	tstatistic
1	"0.38234"^^xsd:decimal

Showing 1 to 1 of 1 entries

Figure 44: One sample T test query result in Apache Jena Fuseki for Chronic Diseases Dataset.

Since T value is less than 1.782, we accept the null hypothesis as this value lies in the acceptance region. Later, the mean of the sample during 2003-2004 is observed to be 10.69286. The difference is not much. Hence our decision of accepting the null hypothesis is a valid one.

This test is repeated in R and the results are as below:



```
> query <- "PREFIX fn:<http://example.org/function#>
+ select (fn:oneTTest(group_concat(?o;separator=','),10.10714,13) as ?tstatistic)
+ where{
+ ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/indicator> \"Prevalence of major cardiovascular disease among US adults
 (20+); NHANES\" .
+ ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/year> \"2003-2004\" .
+ ?s <https://chronicdata.cdc.gov/resource/_5svk-8bnq/data_value> ?o
+ }";
> qd <- SPARQL(endpoint,query);
> qd
$results
  tstatistic
1    0.38234
```

Figure 45: One sample T test calculated in R for Chronic Diseases dataset.

Query 21: Two sample upper tailed T test

Hypothesis: A researcher believes that mean smoking levels in Georgia are higher than those of Missouri. We are considering a sample size of 9 and a confidence percentage of 95.

Null hypothesis: $\mu_1 = \mu_2$

Alternative Hypothesis: $\mu_1 > \mu_2$

α: 0.05

Sample size: 9

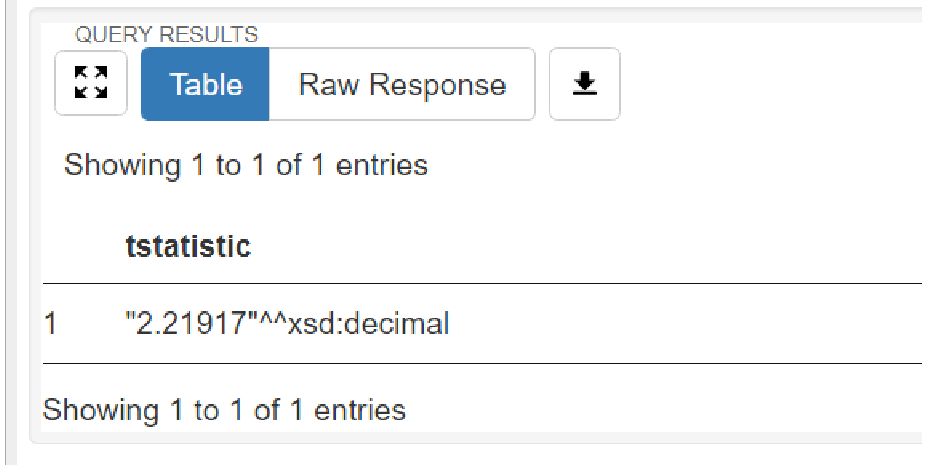
Degrees of freedom: 2*sample size -2 = 16

T score at α=0.025(since it is two sample test) and degrees of freedom 16 is 2.120

Analysis: If the T score is greater than 2.120, we reject the null hypothesis, as the value lies in rejection region.

Query

```
PREFIX fn:<http://example.org/function#>
SELECT (fn:twoTTest(?ogaConcat,?omoConcat,9) AS ?tstatistic)
WHERE{
                 {
                SELECT (group_concat(?omo;separator=':') AS ?omoConcat)
                                WHERE{
                                ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/measuredesc>
"Smoking Status" .
                          ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/locationabbr>
"MO" .
                          ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/data_value> ?omo
.
                          FILTER(regex(?omo,'[0-9.]'))
                                }
                  }
                  {
                SELECT (group_concat(?oga;separator=':') AS ?ogaConcat)
                WHERE{
                           ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/measuredesc>
"Smoking Status" .
                           ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/locationabbr>
"GA" .
                           ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/data_value>
?oga .
                           FILTER(regex(?oga,'[0-9.]'))
                       }
                }
                }
```


QUERY RESULTS

Showing 1 to 1 of 1 entries

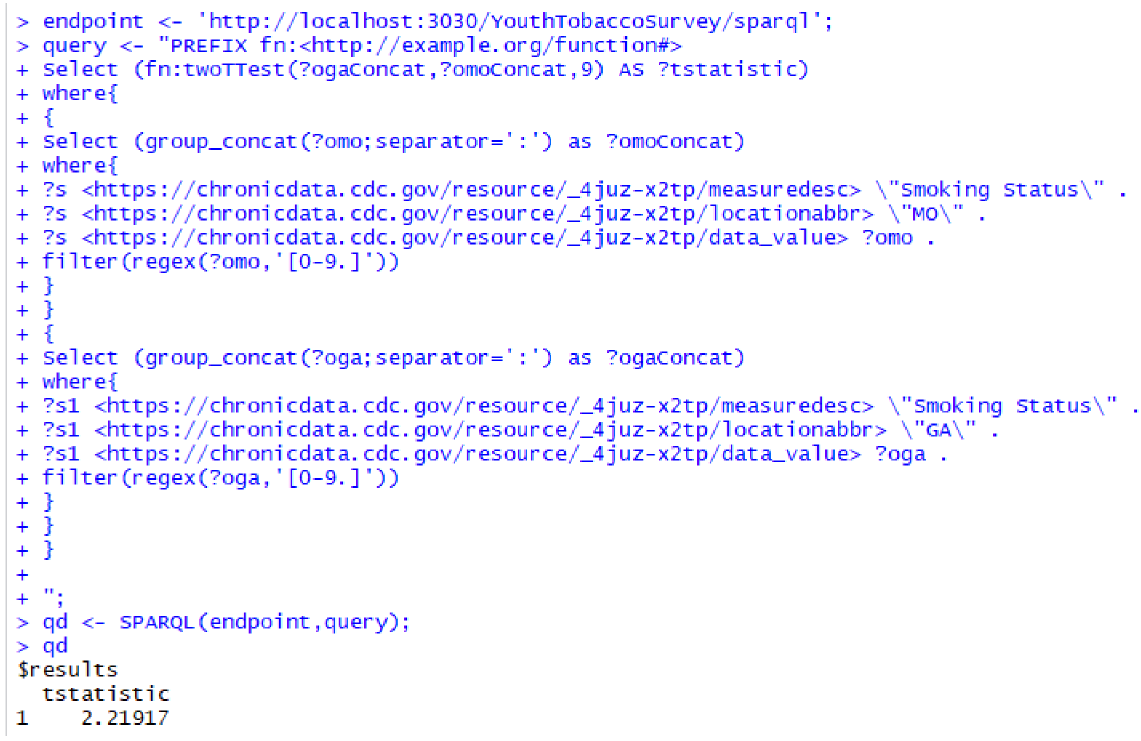
```
     tstatistic
1    "2.21917"^^xsd:decimal
```

Showing 1 to 1 of 1 entries

Figure 46: Two Samples T Test query result in Apache Jena Fuseki for Youth Tobacco Survey dataset.

Since, the resultant T score is greater than 2.120 from the T distribution table, the value lies in the rejection region. Hence, we reject null hypothesis and conclude that the Georgia has greater smoking levels than those of Missouri.

The same test is repeated in R and below figure has the resultant T score same as in SPARQL.



```
> endpoint <- 'http://localhost:3030/YouthTobaccoSurvey/sparql';
> query <- "PREFIX fn:<http://example.org/function#>
+ Select (fn:twoTTest(?ogaConcat,?omoConcat,9) AS ?tstatistic)
+ where{
+ {
+ Select (group_concat(?omo;separator=':') as ?omoConcat)
+ where{
+ ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/measuredesc> \"Smoking Status\" .
+ ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/locationabbr> \"MO\" .
+ ?s <https://chronicdata.cdc.gov/resource/_4juz-x2tp/data_value> ?omo .
+ filter(regex(?omo,'[0-9.]'))
+ }
+ }
+ {
+ Select (group_concat(?oga;separator=':') as ?ogaConcat)
+ where{
+ ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/measuredesc> \"Smoking Status\" .
+ ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/locationabbr> \"GA\" .
+ ?s1 <https://chronicdata.cdc.gov/resource/_4juz-x2tp/data_value> ?oga .
+ filter(regex(?oga,'[0-9.]'))
+ }
+ }
+ }
+
+ ";
> qd <- SPARQL(endpoint,query);
> qd
$results
  tstatistic
1    2.21917
```

Figure 47: Two Samples T Test Statistic calculated in R for Youth Tobacco Survey dataset.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this work, we implemented a library that allows users to query statistical data in RDF using SPARQL. When the library is added in the user projects, it allows users to query mean, median, standard deviation, variance, and skewness and also performs advanced statistical analytics like correlation testing and hypothesis testing. This system eliminates the need to transfer RDF data to other computation environments to run analytics. However, this system requires the users to have knowledge on writing SPARQL queries. Aggregation queries like mean, median, standard deviation, variance, and skewness work similar to other aggregation functions like max, min etc., available in the standard SPARQL queries. However, for writing queries related to hypothesis testing and correlation analysis, it requires the user to understand how such functions should be written.

There are other systems that offer statistical querying capabilities for RDF data. However, they all involve exporting the RDF data or other environments, modifying the content as per native data structures, and running the statistical functions available in the target environments. In contrast, our system adds support for native queries by adding required dependencies and evaluation of statistical functions as part of the SPARQL function library. Also, this can be extended to other Linked Open Data applications where in data can be consumed as streaming data.

At present, this library works with Apache Jena alone. In the future, this can be extended to other SPARQL query processing environments such as OpenLink Virtuoso [25] and Blazegraph [26]. Also, at present, the system requires users to have knowledge on how to execute hypothesis testing and correlation testing queries and this runs in multiple steps. There is a scope for improving these queries depending on future specifications in SPARQL. If SPAQL specification allows for usage of List data structures in queries, these queries related to hypothesis testing and correlation testing can be improved. Also, the capabilities to run text analytics on RDF data can be added to the system in a similar fashion, i.e., the way we are utilizing R environment. Thus, our present work can be extended as a complex library that supports all forms of analytics for RDF data in SPARQL utilizing R.

REFERENCES

1. W3C SEMANTIC WEB. Available from: https://www.w3.org/2001/sw/ .
2. Spatial searches with SPARQL. Available from: https://jena.apache.org/documentation/query/spatial-query.html
3. SparqlImplementations. Available from: https://www.w3.org/wiki/index.php?title=SparqlImplementations.
4. RichardCyganiak. SPARQL. 2009; Available from: https://www.w3.org/wiki/SPARQL.
5. TIM BERNERS-LEE, James Hendler; Ora Lassila (May 17, 2001)., The Semantic Web, in Scientific American. 2001.
6. Resource Description Framework (RDF). Available from: https://www.w3.org/RDF/.
7. Max Völkel, Y.S., RDFReactor-from ontologies to programmatic data access. Poster Proceedings of the Fourth International Semantic Web Conference, 2005
8. The RDF Data Cube Vocabulary. Available from: https://www.w3.org/TR/vocab-data-cube/
9. Bo Hu, Eduardo Mendes Rodrigues and Emeric Viel. Programmable Analytics for Linked Data. In iiWAS Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services. 2014. New York.
10. Willem Robert van Hage, w.c.f.T.K., Benedikt Graeler, Christopher Davis, Jesper Hoeksema, Alan Ruttenberg, and Daniel Bahls, Package ‘SPARQL’. 2015.

11. Ciro Baron Neto, K.M., Martin Brummer, Dimitris Kontokostas, Sebastian Hellman, LODVader: An Interface to LOD Visualization, Analytics and DiscovERy in Real-time, in WWW '16 Companion Proceedings of the 25th International Conference Companion on World Wide Web. 2016.

12. JRI - Java/R Interface. Available from: http://www.rforge.net/JRI/

13. Jena Full Text Search. Available from: https://jena.apache.org/documentation/query/text-query.html.

14. Functions in ARQ. Available from: https://jena.apache.org/documentation/query/library-function.html

15. Evangelos Kalampokis, A.N., Peter Haase, Richard Cyganiak, Arkadiusz Stasiewicz , Areti Karamanou, Maria Zotou, Dimitris Zeginis, Efthimios Tambouris , Konstantinos Tarabanis. Exploiting linked data cubes with opencube toolkit. in ISWC-PD'14 Proceedings of the 2014 International Conference on Posters & Demonstrations Track - Volume 1272. 2014. Riva del Garda, Italy.

16. ARQ - A SPARQL Processor for Jena. Available from: https://jena.apache.org/documentation/query/.

17. SPSS Inc., S., SPSS for Windows. 2007: Chicago.

18. Identifying Resources. SPARQL Query Tests; Available from: https://www.w3.org/2001/sw/DataAccess/rq23/examples.html

19. Lee Feigenbaum, SPARQL By Example. 2009-06-09; Available from: https://www.w3.org/2009/Talks/0615-qbe/

20. Bob DuCharme, B. Appreciating SPARQL CONSTRUCT more. 9 September 2009; Available from: http://www.snee.com/bobdc.blog/2009/09/appreciating-sparql-construct.html.

21. Asking "yes or no" questions. SPARQL Query Tests; Available from: https://www.w3.org/2001/sw/DataAccess/rq23/examples.html.

22. SAS Institute Inc., SAS 9.1.3 Help and Documentation 2002-2004, SAS Institute Inc.,.

23. SPSS Inc., S., SPSS for Windows. 2007: Chicago.

24. Andy Seaborne, A.P., Lee Feigenbaum, Gregory Todd Williams. SPARQL 1.1 Federated Query. 2013; Available from: https://www.w3.org/TR/sparql11-federated-query/.

25. OpenLink Virtuoso, O. OpenLink Virtuoso Universal Server Documentation. 1999-2018; Available from: http://docs.openlinksw.com/virtuoso/.

26. Blazegraph. 2015; Available from: https://wiki.blazegraph.com/wiki/index.php/Main_Page.