INDEX

1. Multidimensional Data Models

Aim: Understand and write a program to implement the Star Schema, Snowflake Schema, and Fact Constellation Schema.

Description:

Star Schema:

- A single large central fact table and one table for each dimension
- Every fact points to one tuple in each of the dimensions and has additional attributes
- Does not capture hierarchies directly

Snowflake Schema:

- Variant of star schema model
- A single, large and central fact table and one or more tables for each dimension
- Dimension tables are normalized split dimension table data into additional tables

Fact Constellation:

- Multiple fact tables share dimension tables
- This schema is viewed as collection of stars hence called galaxy schema or fact constellation
- Sophisticated applications require such schema

a. Star Schema

Program:

```
class DimLocation:
    def __init__(self, location_id, street, city, state, country):
        self.location_id = location_id
        self.street = street
        self.city = city
        self.state = state
        self.country = country

class DimProduct:
    def __init__(self, product_id, name, cost_price, selling_price):
        self.product_id = product_id
        self.name = name
        self.cost_price = cost_price
        self.selling_price = selling_price

class FactSales:
    def __init__(self, location_id, product_id, units_sold, rupees_sold):
        self.location_id = location_id
        self.product_id = product_id
        self.units_sold = units_sold
        self.rupees_sold = rupees_sold
```

```
def main():
    print('----------DimLocation Table----------\n')
    locations = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nLocation '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for location in locations:
                if location.location_id == id:
                    print("\tID already exists!")
                    loop = True
                    break
            if not loop:
                break

        street = input("\tStreet: ")
        city = input("\tCity: ")
        state = input("\tState: ")
        country = input("\tCountry: ")
        locations.append(DimLocation(id, street, city, state, country))

    print('\n----------DimProduct Table----------\n')
    products = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nProduct '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for product in products:
                if product.product_id == id:
                    print("\tID already exists!")
                    loop = True
                    break
            if not loop:
                break
        name = input("\tName: ")
        cost_price = input("\tCost Price: ")
        selling_price = input("\tSelling Price: ")
        products.append(DimProduct(id, name, cost_price, selling_price))

    print('\n----------FactSales Table----------\n')
    sales = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nSale '+str(i+1)+':')
        while True:
            location_id = input("\tLocation ID: ")
            loop = True
            for location in locations:
                if location.location_id == location_id:
                    loop = False
                    break
            if not loop:
                break
            else:
                print("\tNo such DimLocation entry found!")
        while True:
```

```
        product_id = input("\tProduct ID: ")
        loop = True
        for product in products:
            if product.product_id == product_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimProduct entry found!")
    units_sold = input("\tUnits Sold: ")
    rupees_sold = input("\tRupees Sold: ")
    sales.append(FactSales(location_id, product_id, units_sold, rupees_sold))

print("\n\nDimLocation Entries:")
for location in locations:

print(location.location_id+"\t"+location.street+"\t\t\t"+location.city+"\t\t\t"+location.state+"\t\t\t"+location.count
ry)
print("\nDimProduct Entries:")
for product in products:
    print(product.product_id+"\t"+product.name+"\t\t\t"+product.cost_price+"\t"+product.selling_price)
print("\nFactSales Entries:")
for sale in sales:
    print(sale.location_id+"\t"+sale.product_id+"\t"+sale.units_sold+"\t"+sale.rupees_sold)

if __name__ == "__main__":
    main()
```

Input & Output:

----------DimLocation Table----------

Number of Entries: 2

Location 1:
    ID: 1
    Street: VST Colony
    City: Hyderabad
    State: Telangana
    Country: India

Location 2:
    ID: 2
    Street: HMT Nagar
    City: Hyderabad
    State: Telangana
    Country: India

----------DimProduct Table----------

Number of Entries: 2

Product 1:
    ID: 1

```
        Name: Surface 2
        Cost Price: 150000
        Selling Price: 153000

Product 2:
        ID: 2
        Name: Pixel 2
        Cost Price: 55000
        Selling Price: 60000

----------FactSales Table----------

Number of Entries: 2

Sale 1:
        Location ID: 1
        Product ID: 2
        Units Sold: 4
        Rupees Sold: 240000

Sale 2:
        Location ID: 2
        Product ID: 1
        Units Sold: 2
        Rupees Sold: 306000


DimLocation Entries:
1       VST Colony              Hyderabad               Telangana
                India
2       HMT Nagar               Hyderabad               Telangana
                India

DimProduct Entries:
1       Surface 2       150000  153000
2       Pixel 2          55000   60000

FactSales Entries:
1       2       4       240000
2       1       2       306000
```

b. Snowflake Schema

Program:

```
class DimCity:
    def __init__(self, city_id, city, state, country):
        self.city_id = city_id
        self.city = city
        self.state = state
        self.country = country
```

```
class DimLocation:
    def __init__(self, location_id, street, city_id):
        self.location_id = location_id
        self.street = street
        self.city_id = city_id

class DimProduct:
    def __init__(self, product_id, name, cost_price, selling_price):
        self.product_id = product_id
        self.name = name
        self.cost_price = cost_price
        self.selling_price = selling_price

class FactSales:
    def __init__(self, location_id, product_id, units_sold, rupees_sold):
        self.location_id = location_id
        self.product_id = product_id
        self.units_sold = units_sold
        self.rupees_sold = rupees_sold

def main():
    print('----------DimCity Table----------\n')
    cities = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nCity '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for city in cities:
                if city.city_id == id:
                    print("\tID already exists!")
                    loop = True
                    break
            if not loop:
                break
        city = input("\tCity: ")
        state = input("\tState: ")
        country = input("\tCountry: ")
        cities.append(DimCity(id, city, state, country))

    print('\n----------DimLocation Table----------\n')
    locations = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nLocation '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for location in locations:
                if location.location_id == id:
                    print("\tID already exists!")
                    loop = True
                    break
            if not loop:
                break
        street = input("\tStreet: ")
        city_id = input("\tCity ID: ")
        locations.append(DimLocation(id, street, city_id))
```

```
print('\n----------DimProduct Table----------\n')
products = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nProduct '+str(i+1)+':')
    while True:
        id = input("\tID: ")
        loop = False
        for product in products:
            if product.product_id == id:
                print("\tID already exists!")
                loop = True
                break
        if not loop:
            break
    name = input("\tName: ")
    cost_price = input("\tCost Price: ")
    selling_price = input("\tSelling Price: ")
    products.append(DimProduct(id, name, cost_price, selling_price))

print('\n----------FactSales Table----------\n')
sales = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nSale '+str(i+1)+':')
    while True:
        location_id = input("\tLocation ID: ")
        loop = True
        for location in locations:
            if location.location_id == location_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimLocation entry found!")
    while True:
        product_id = input("\tProduct ID: ")
        loop = True
        for product in products:
            if product.product_id == product_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimProduct entry found!")
    units_sold = input("\tUnits Sold: ")
    rupees_sold = input("\tRupees Sold: ")
    sales.append(FactSales(location_id, product_id, units_sold, rupees_sold))

print("\n\nDimCity Entries:")
for city in cities:
    print(city.city_id+"\t"+city.city+"\t\t\t"+city.state+"\t\t\t"+city.country)
print("\n\nDimLocation Entries:")
for location in locations:
    print(location.location_id+"\t"+location.street+"\t\t\t"+location.city_id)
print("\nDimProduct Entries:")
for product in products:
    print(product.product_id+"\t"+product.name+"\t\t\t"+product.cost_price+"\t"+product.selling_price)
```

```
    print("\nFactSales Entries:")
    for sale in sales:
        print(sale.location_id+"\t"+sale.product_id+"\t"+sale.units_sold+"\t"+sale.rupees_sold)

if __name__ == "__main__":
    main()
```

Input & Output:

```
----------DimCity Table----------

Number of Entries: 1

City 1:
        ID: 1
        City: Hyderabad
        State: Telangana
        Country: India

----------DimLocation Table----------

Number of Entries: 2

Location 1:
        ID: 1
        Street: VST Colony
        City ID: 1

Location 2:
        ID: 2
        Street: HMT Nagar
        City ID: 2

----------DimProduct Table----------

Number of Entries: 2

Product 1:
        ID: 1
        Name: Surface 2
        Cost Price: 150000
        Selling Price: 153000

Product 2:
        ID: 2
        Name: Pixel 2
        Cost Price: 55000
        Selling Price: 60000

----------FactSales Table----------
```

Number of Entries: 2

Sale 1:
    Location ID: 1
    Product ID: 2
    Units Sold: 4
    Rupees Sold: 240000

Sale 2:
    Location ID: 2
    Product ID: 1
    Units Sold: 2
    Rupees Sold: 306000

DimCity Entries:
1   Hyderabad   Telangana   India

DimLocation Entries:
1   VST Colony   1
2   HMT Nagar   2

DimProduct Entries:
1   Surface 2   150000 153000
2   Pixel 2   55000   60000

FactSales Entries:
1   2   4   240000
2   1   2   306000

c. Fact Constellation Schema

Program:

```
class DimDate:
    def __init__(self, date_id, date):
        self.date_id = date_id
        self.date = date

class DimProduct:
    def __init__(self, product_id, name, cost_price, selling_price):
        self.product_id = product_id
        self.name = name
        self.cost_price = cost_price
        self.selling_price = selling_price

class DimArea:
    def __init__(self, area_id, street, city, state, country):
        self.area_id = area_id
        self.street = street
        self.city = city
        self.state = state
        self.country = country
```

```
class DimRetailer:
    def __init__(self, retailer_id, name, area_id):
        self.retailer_id = retailer_id
        self.name = name
        self.area_id = area_id

class DimSupplier:
    def __init__(self, supplier_id, name, area_id):
        self.supplier_id = supplier_id
        self.name = name
        self.area_id = area_id

class FactPurchases:
    def __init__(self, product_id, date_id, supplier_id, buyer_id, units_purchased):
        self.product_id = product_id
        self.date_id = date_id
        self.supplier_id = supplier_id
        self.buyer_id = buyer_id
        self.units_purchased = units_purchased

class FactSales:
    def __init__(self, product_id, date_id, retailer_id, units_sold):
        self.product_id = product_id
        self.date_id = date_id
        self.retailer_id = retailer_id
        self.units_sold = units_sold

def main():
    print('----------DimDate Table----------\n')
    dates = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nDate '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for date in dates:
                if date.date_id == id:
                    print("\tID already exists!")
                    loop = True
                    break
            if not loop:
                break

        date = input("\tDate: ")
        dates.append(DimDate(id, date))

    print('\n----------DimProduct Table----------\n')
    products = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nProduct '+str(i+1)+':')
        while True:
            id = input("\tID: ")
            loop = False
            for product in products:
                if product.product_id == id:
                    print("\tID already exists!")
                    loop = True
```

```
                break
        if not loop:
            break
    name = input("\tName: ")
    cost_price = input("\tCost Price: ")
    selling_price = input("\tSelling Price: ")
    products.append(DimProduct(id, name, cost_price, selling_price))

print('\n----------DimArea Table----------\n')
areas = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nArea '+str(i+1)+':')
    while True:
        id = input("\tID: ")
        loop = False
        for area in areas:
            if area.area_id == id:
                print("\tID already exists!")
                loop = True
                break
        if not loop:
            break
    street = input("\tStreet: ")
    city = input("\tCity: ")
    state = input("\tState: ")
    country = input("\tCountry: ")
    areas.append(DimArea(id, street, city, state, country))

print('\n----------DimSupplier Table----------\n')
suppliers = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nSupplier '+str(i+1)+':')
    while True:
        id = input("\tID: ")
        loop = False
        for supplier in suppliers:
            if supplier.supplier_id == id:
                print("\tID already exists!")
                loop = True
                break
        if not loop:
            break
    name = input("\tName: ")
    area_id = input("\tArea ID: ")
    suppliers.append(DimSupplier(id, name, area_id))

print('\n----------DimRetailer Table----------\n')
retailers = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nRetailer '+str(i+1)+':')
    while True:
        id = input("\tID: ")
        loop = False
        for retailer in retailers:
            if retailer.retailer_id == id:
                print("\tID already exists!")
                loop = True
```

```
                break
        if not loop:
            break
    name = input("\tName: ")
    area_id = input("\tArea ID: ")
    retailers.append(DimRetailer(id, name, area_id))

print('\n----------FactPurchases Table----------\n')
purchases = []
entries = int(input('Number of Entries: '))
for i in range(entries):
    print('\nSale '+str(i+1)+':')
    while True:
        product_id = input("\tProduct ID: ")
        loop = True
        for product in products:
            if product.product_id == product_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimProduct entry found!")
    while True:
        date_id = input("\tDate ID: ")
        loop = True
        for date in dates:
            if date.date_id == date_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimDate entry found!")
    while True:
        supplier_id = input("\tSupplier ID: ")
        loop = True
        for supplier in suppliers:
            if supplier.supplier_id == supplier_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimSupplier entry found!")
    while True:
        buyer_id = input("\tBuyer ID: ")
        loop = True
        for buyer in retailers:
            if buyer.retailer_id == buyer_id:
                loop = False
                break
        if not loop:
            break
        else:
            print("\tNo such DimRetailer entry found!")
    units_purchased = input("\tUnits Purchased: ")
    purchases.append(FactPurchases(product_id, date_id, supplier_id, buyer_id, units_purchased))

print('\n----------FactSales Table----------\n')
```

```
    sales = []
    entries = int(input('Number of Entries: '))
    for i in range(entries):
        print('\nSale '+str(i+1)+':')
        while True:
            product_id = input("\tProduct ID: ")
            loop = True
            for product in products:
                if product.product_id == product_id:
                    loop = False
                    break
            if not loop:
                break
            else:
                print("\tNo such DimProduct entry found!")
        while True:
            date_id = input("\tDate ID: ")
            loop = True
            for date in dates:
                if date.date_id == date_id:
                    loop = False
                    break
            if not loop:
                break
            else:
                print("\tNo such DimDate entry found!")
        while True:
            retailer_id = input("\tRetailer ID: ")
            loop = True
            for retailer in retailers:
                if retailer.retailer_id == retailer_id:
                    loop = False
                    break
            if not loop:
                break
            else:
                print("\tNo such DimRetailer entry found!")
        units_sold = input("\tUnits Sold: ")
        sales.append(FactSales(product_id, date_id, retailer_id, units_sold))

    print("\n\nDimDate Entries:")
    for date in dates:
        print('{0:5} {1:10}'.format(date.date_id, date.date))
    print("\nDimProduct Entries:")
    for product in products:
        print('{0:5} {1:15} {2:10} {3:10}'.format(product.product_id, product.name, product.cost_price,
product.selling_price))
    print("\nDimArea Entries:")
    for area in areas:
        print('{0:5} {1:15} {2:15} {3:15} {4:15}'.format(area.area_id, area.street, area.city, area.state,
area.country))
    print("\nDimRetailer Entries:")
    for retailer in retailers:
        print('{0:5} {1:20} {2:5}'.format(retailer.retailer_id, retailer.name, retailer.area_id))
    print("\nDimSupplier Entries:")
    for supplier in suppliers:
        print('{0:5} {1:15} {2:5}'.format(supplier.supplier_id, supplier.name, supplier.area_id))
    print("\nFactPuchases Entries:")
    for purchase in purchases:
```

```
        print('{0:5} {1:5} {2:5} {3:5} {4:10}'.format(purchase.product_id, purchase.date_id, purchase.supplier_id,
purchase.buyer_id, purchase.units_purchased))
    print("\nFactSales Entries:")
    for sale in sales:
        print('{0:5} {1:5} {2:5} {3:10}'.format(sale.product_id, sale.date_id, sale.retailer_id, sale.units_sold))

if __name__ == "__main__":
    main()
```

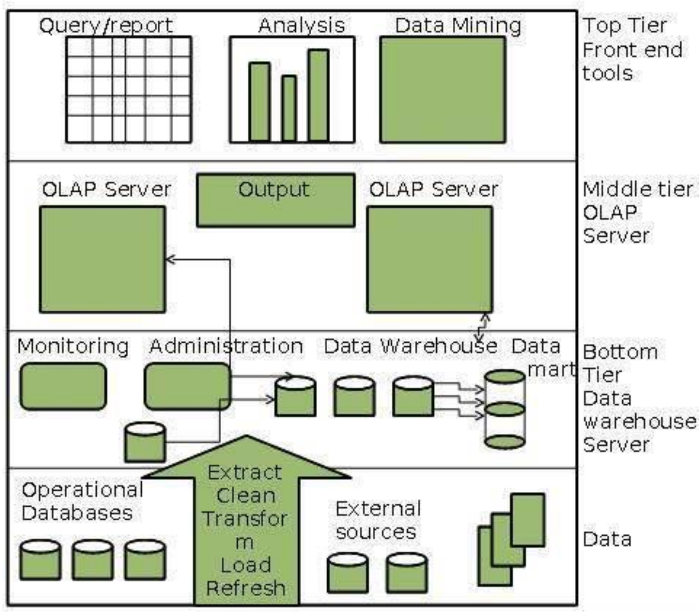
Input & Output:

```
----------DimDate Table----------

Number of Entries: 1

Date 1:
        ID: 1
        Date: 18/09/2018

----------DimProduct Table----------

Number of Entries: 1

Product 1:
        ID: 1
        Name: Surface 2
        Cost Price: 150000
        Selling Price: 153000

----------DimArea Table----------

Number of Entries: 1

Area 1:
        ID: 1
        Street: VST Colony
        City: Hyderabad
        State: Telangana
        Country: India

----------DimSupplier Table----------

Number of Entries: 1

Supplier 1:
        ID: 1
        Name: ABC Co.
        Area ID: 1

----------DimRetailer Table----------

Number of Entries: 1
```

```
Retailer 1:
	ID: 1
	Name: Reliance Digital
	Area ID: 1

----------FactPurchases Table----------

Number of Entries: 1

Sale 1:
	Product ID: 1
	Date ID: 1
	Supplier ID: 1
	Buyer ID: 1
	Units Purchased: 4

----------FactSales Table----------

Number of Entries: 1

Sale 1:
	Product ID: 1
	Date ID: 1
	Retailer ID: 1
	Units Sold: 2


DimDate Entries:
1	18/09/2018

DimProduct Entries:
1	Surface 2	150000	153000

DimArea Entries:
1	VST Colony	Hyderabad	Telangana	India

DimRetailer Entries:
1	Reliance Digital	1

DimSupplier Entries:
1	ABC Co.		1

FactPuchases Entries:
1	1	1	1	4

FactSales Entries:
1	1	1	2
```

2. Data Warehousing

Three-Tier Data Warehouse Architecture



- **Bottom Tier** - The bottom tier of the architecture is the data warehouse database server. It is the relational database system. We use the back-end tools and utilities to feed data into the bottom tier. These back-end tools and utilities perform the Extract, Clean, Load, and refresh functions.

- **Middle Tier** - In the middle tier, we have the OLAP Server that can be implemented in either of the following ways.

    - By Relational OLAP (ROLAP), which is an extended relational database management system. The ROLAP maps the operations on multidimensional data to standard relational operations.

    - By Multidimensional OLAP (MOLAP) model, which directly implements the multidimensional data and operations.

- **Top-Tier** - This tier is the front-end client layer. This layer holds the query tools and reporting tools, analysis tools and data mining tools.

3. OLAP Cube

An **OLAP cube** is a term that typically refers to multi-dimensional array of data. *OLAP* is an acronym for online analytical processing,[1]which is a computer-based technique of analyzing data to look for insights. The term *cube* here refers to a multi-dimensional dataset, which is also sometimes called a hypercube if the number of dimensions is greater than 3.

Operations:

1. Slice is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension.[4] The picture shows a slicing operation: The sales figures of all sales regions and all product categories of the company in the year 2005 and 2006 are "sliced" out of the data cube.
2. Dice: The dice operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions.[5]The picture shows a dicing operation: The new cube shows the sales figures of a limited number of product categories, the time and region dimensions cover the same range as before.
3. Drill Down/Up allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).[4] The picture shows a drill-down operation: The analyst moves from the summary category "Outdoor-Schutzausrüstung" to see the sales figures for the individual products.
4. Roll-up: A roll-up involves summarizing the data along a dimension. The summarization rule might be computing totals along a hierarchy or applying a set of formulas such as profit = sales – expenses.
5. Pivot: allows an analyst to rotate the cube in space to see its various faces. For example, cities could be arranged vertically and products horizontally while viewing data for a particular quarter. Pivoting could replace products with time periods to see data across time for a single product.

OLAP Server Architectures:

- Relational OLAP (ROLAP) servers: These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use relational or extended relational DBMS to store and manage warehouse data.

- Multidimensional OLAP (MOLAP) servers: These support multidimensional data views through array-based search engines. They map multidimensional views directly to data cube array structures.
- Hybrid OLAP (HOLAP) servers: This approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and faster computation of MOLAP.

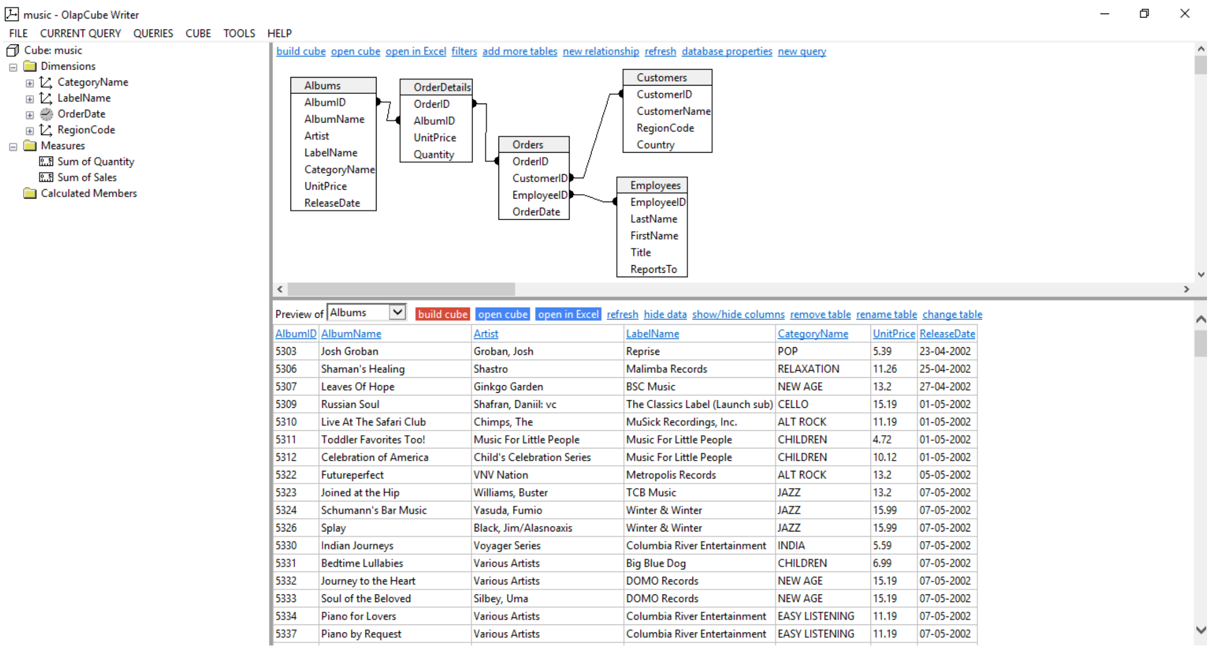


Figure 1: Music Database in OLAP Cube Writer

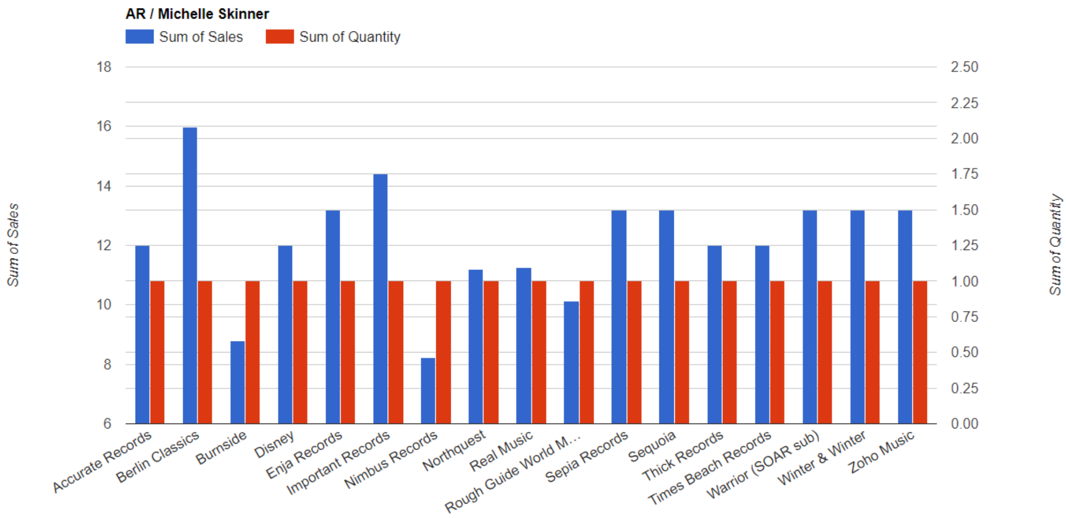


Figure 2: Sum of Sales & Quantities vs Label Name

4. Basics of WEKA tool

Aim:

a. Investigation the Application interfaces of the Weka tool
b. Explore the default datasets

Description: Weka is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization.

a. Investigation the Application interfaces of the Weka tool

Figure 3: Weka Home Screen

Open the program. Once the program has been loaded on the user's machine it is opened by navigating to the programs start option and that will depend on the user's operating system. Figure 3 is an example of the initial opening screen on a computer.

There are five options available on this initial screen:

1. **Explorer** - The graphical interface used to conduct experimentation on raw data. There are six tabs:

   i. Preprocess- Used to choose the data file to be used by the application.

   - **Open File** - Allows for the user to select files residing on the local machine or recorded medium
   - **Open URL** - Provides a mechanism to locate a file or data source from a different location specified by the user
   - **Open Database** - Allows the user to retrieve files or data from a database source provided by user

ii. **Classify -** Used to test and train different learning schemes on the preprocessed data file under experimentation Again there are several options to be selected inside of the classify tab. Test option gives the user the choice of using four different test mode scenarios on the data set.

1. Use training set
2. Supplied training set
3. Cross validation
4. Split percentage

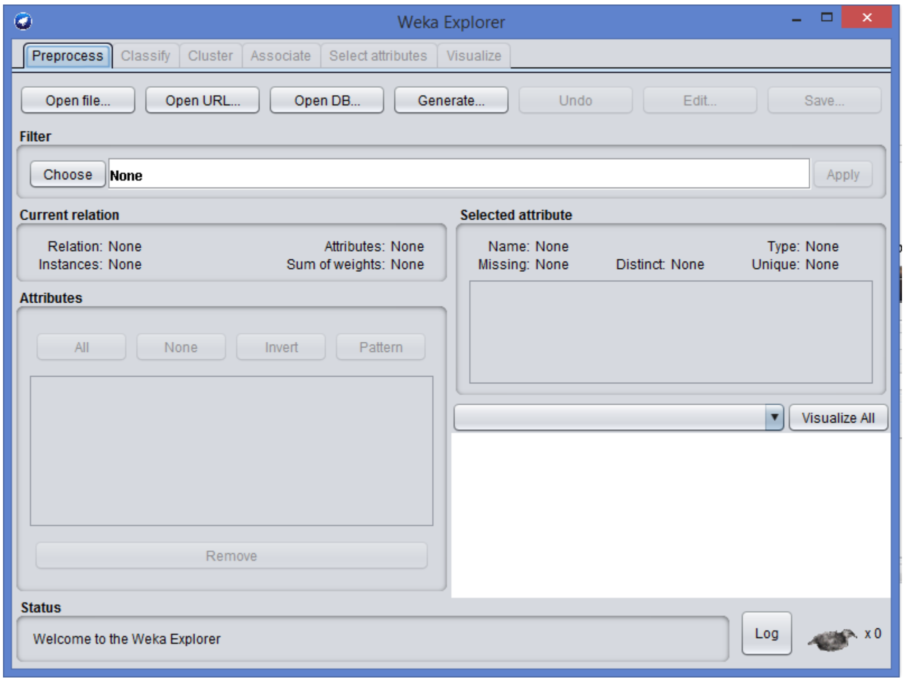


Figure 4: Weka Explorer

iii. Cluster - Used to apply different tools that identify clusters within the data file.
The Cluster tab opens the process that is used to identify commonalties or clusters of occurrences within the data set and produce information for the user to analyze.

iv. Association - Used to apply different rules to the data file that identify association within the data. The associate tab opens a window to select the options for associations within the data set.

v. Select attributes - Used to apply different rules to reveal changes based on selected attributes inclusion or exclusion from the experiment

vi. Visualize - Used to see what the various manipulation produced on the data set in a 2D format, in scatter plot and bar graph output.

2. Experimenter - This option allows users to conduct different experimental variations on data sets and perform statistical manipulation. The Weka Experiment Environment enables the user to create, run, modify, and analyze experiments in a more convenient manner than is possible when processing the schemes individually. For example, the user can create an experiment that runs several schemes against a series of datasets and then analyze the results to determine if one of the schemes is (statistically) better than the other schemes.

3. Knowledge Flow - Basically the same functionality as Explorer with drag and drop functionality. The advantage of this option is that it supports incremental learning from previous results

4. Workbench - A machine learning workbench is a platform or environment that supports and facilitates a range of machine learning activities reducing or removing the need for multiple tools.

5. Simple CLI - provides users without a graphic interface option the ability to execute commands from a terminal window

b. Explore the default datasets

Click the "Open file…" button to open a data set and double click on the "data" directory. Weka provides a number of small common machine learning datasets that you can use to practice on.

Select the "iris.arff" file to load the Iris dataset

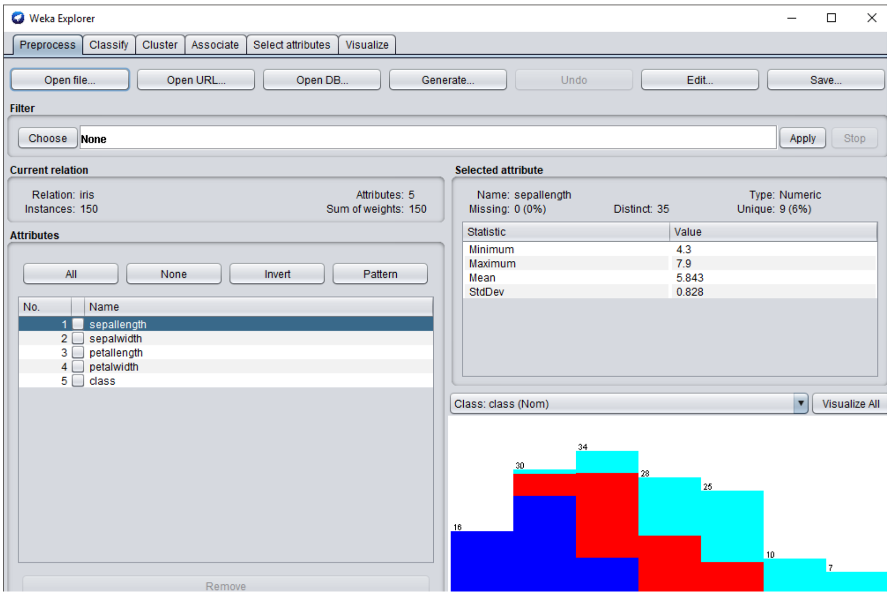


Figure 5: iris.arff

5. Creating new ARFF file

Aim: Creating a new ARFF file

Description: An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software in WEKA. Attribute Relation File Format has two sections:

1) The Header section defines relation (dataset) name, attribute name, and type.
2) The Data section lists the data instances.

```
@relation weather.symbolic

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
overcast,cool,normal,TRUE,yes
sunny,mild,high,FALSE,no
sunny,cool,normal,FALSE,yes
rainy,mild,normal,FALSE,yes
sunny,mild,normal,TRUE,yes
overcast,mild,high,TRUE,yes
overcast,hot,normal,FALSE,yes
rainy,mild,high,TRUE,no
```

Figure 6: ARFF Example

The figure shows an ARFF file for the weather data. Lines beginning with a % sign are comments. And there are three basic keywords:

- "@relation" in Header section, followed with relation name.
- "@attribute" in Header section, followed with attributes name and its type (or range).
- "@data" in Data section, followed with the list of data instances.

Procedure:

1. Create a CSV file in any of the editors (Notepad, Excel etc.)
2. Save it as filename.csv and open Weka.
3. Save it in the .arff format.
4. Open the ARFF file in Notepad to view the relations between attributes (see the above figure for example.)

6. Data Preprocessing Techniques

Aim:

a. Attribute Selection
b. Handling Missing Values
c. Discretization
d. Normalization

Description: Why preprocessing?

Real world data are generally

- Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- Noisy: containing errors or outliers
- Inconsistent: containing discrepancies in codes or names

Tasks in data preprocessing

- Data cleaning: fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies.
- Data integration: using multiple databases, data cubes, or files.
- Data transformation: normalization and aggregation.
- Data reduction: reducing the volume but producing the same or similar analytical results.
- Data discretization: part of data reduction, replacing numerical attributes with nominal ones.

a. Attribute selection

Attribute selection is also called variable selection or feature selection. It is the automatic selection of attributes in your data that are most relevant to the predictive modeling problem you are working on.

1. To search through all possible combinations of attributes in the data and find which subset of attributes works best for prediction, make sure that you set up attribute evaluator to 'Cfs SubsetEval' and a search method to 'Best First'. The evaluator will determine what method to use to assign a worth to each subset of attributes. The search method will determine what style of search to perform.

i. Use full training set. The worth of the attribute subset is determined using the full set of training data.

ii. Cross-validation. The worth of the attribute subset is determined by a process of cross-validation. The 'Fold' and 'Seed' fields set the number of folds to use and the random seed used when shuffling the data. Specify which attribute to treat as the class in the drop-down box below the test options. Once all the test options are set, you can start the attribute selection process by clicking on 'Start' button. When it is finished, the results of selection are shown on the right part of the window and entry is added to the 'Result list'.

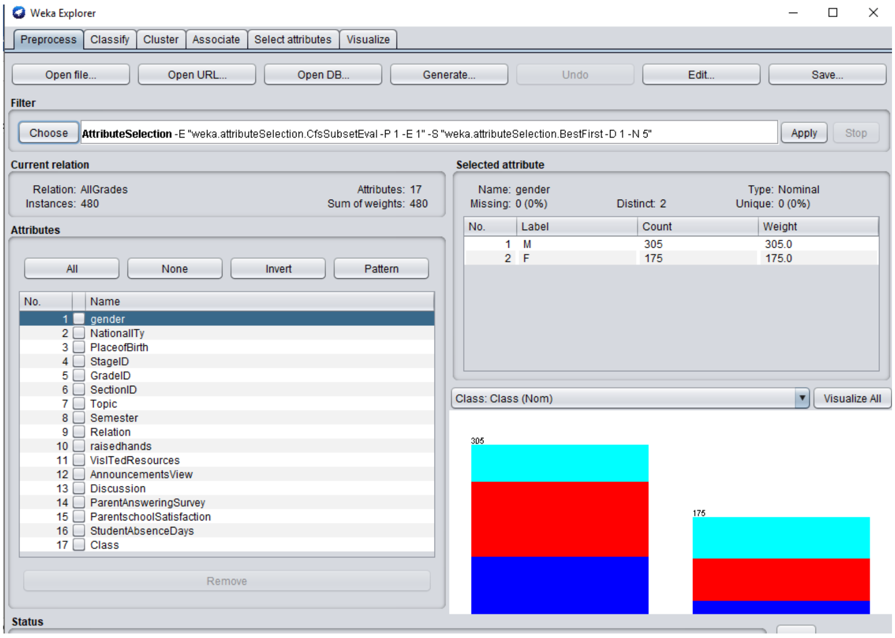


Figure 7: students.arff (Before applying Attribute Selection)

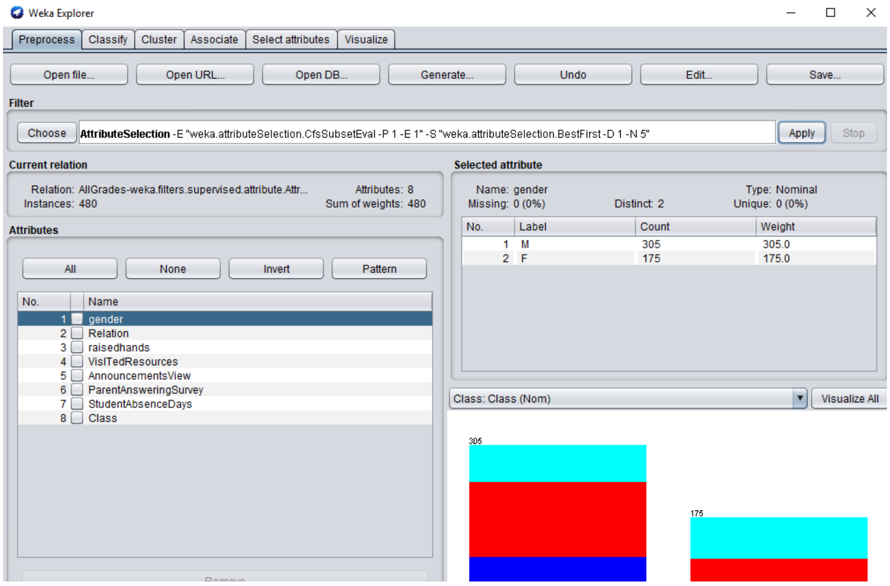


Figure 8: students.arff (After applying Attribute Selection)

2. Visualizing results

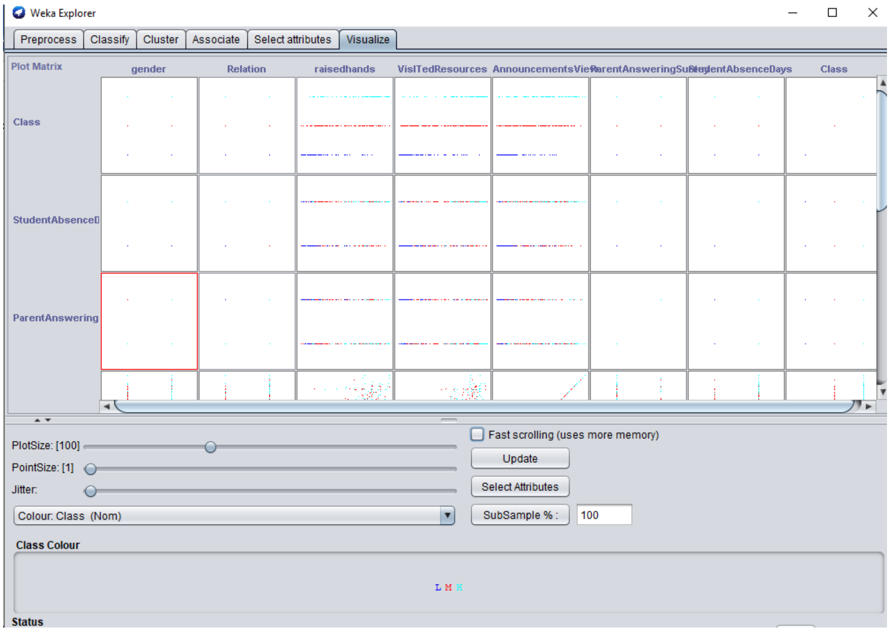


Figure 9: Visualize

b. Handling missing values

By attribute mean: This method is used for data sets with numerical attributes. In this method, every missing attribute value for a numerical attribute is replaced by the arithmetic mean of known attribute values.

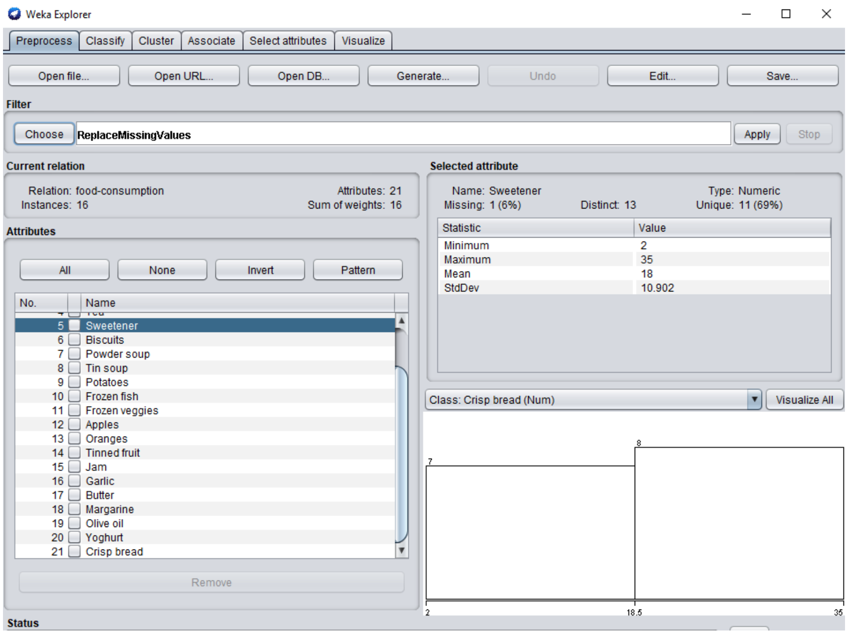


Figure 10: food-consumption.csv (Before handling missing values)

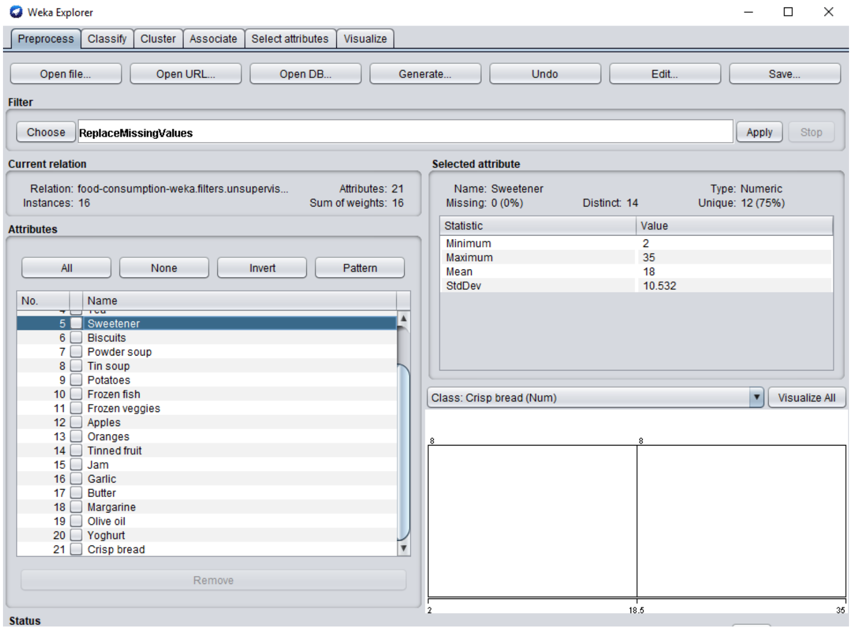


Figure 11: food-consumption.csv (After applying ReplaceMissingValues filter)

c. Discretization

Discretization is the process of converting continuous attribute to discrete attribute. It is a process of data reduction. Data discretization transforms numeric data by mapping values to interval or concept labels. Such methods can be used to automatically generate concept hierarchies for the data, which allows for mining at multiple levels of granularity. Discretization techniques include binning, histogram analysis, cluster analysis, decision tree analysis, and correlation analysis. For nominal data, concept hierarchies may be generated based on schema definitions as well as the number of distinct values per attribute.

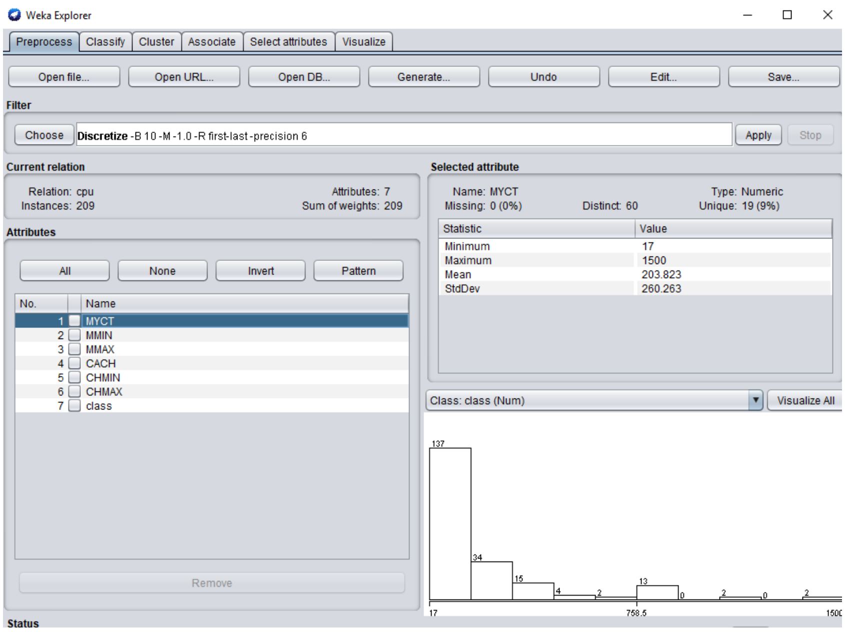


Figure 12: cpu.arff (Before discretization)

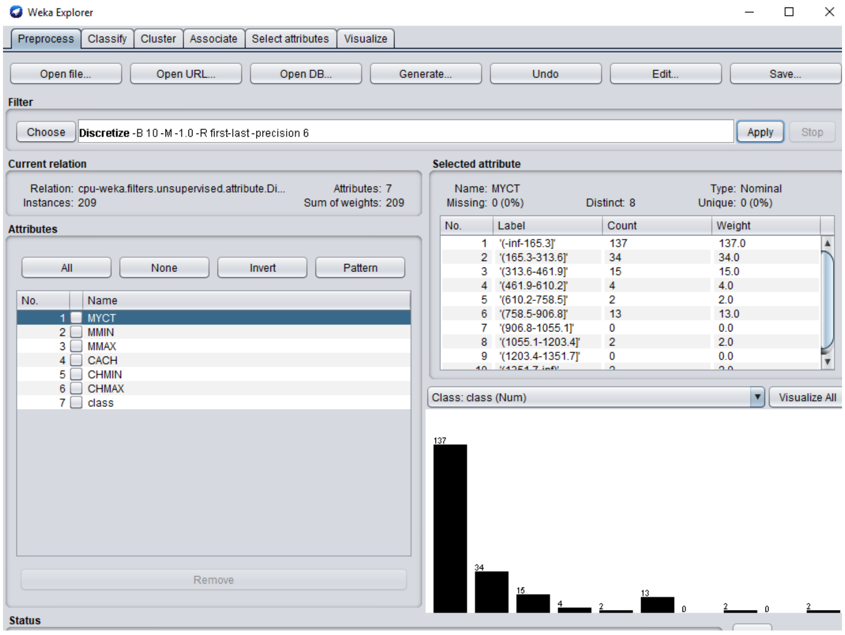


Figure 13: cpu.arff (After discretization)

d. Normalization

Normalization is the process of converting the scale of an attribute from one form to other form. Normalization is of 3 types:

- Min-max Normalization: Min max normalization performs a linear transformation on the original data.
- Z score normalization (zero-mean normalization): The values for an attribute, A,are normalized based on the mean(i.e., average) and standard deviation of A.
- Decimal scaling: Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A.

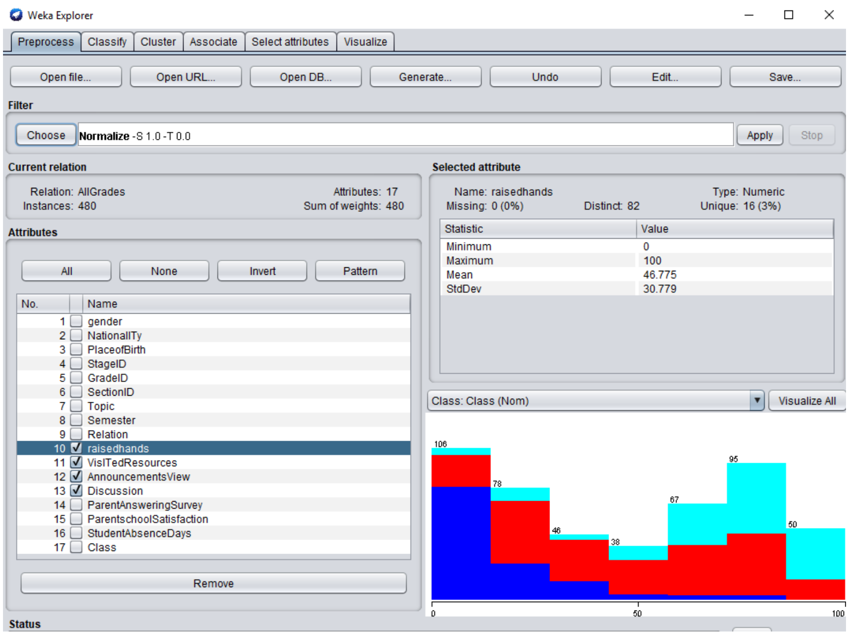


Figure 14: students.arff (Before normalizing)

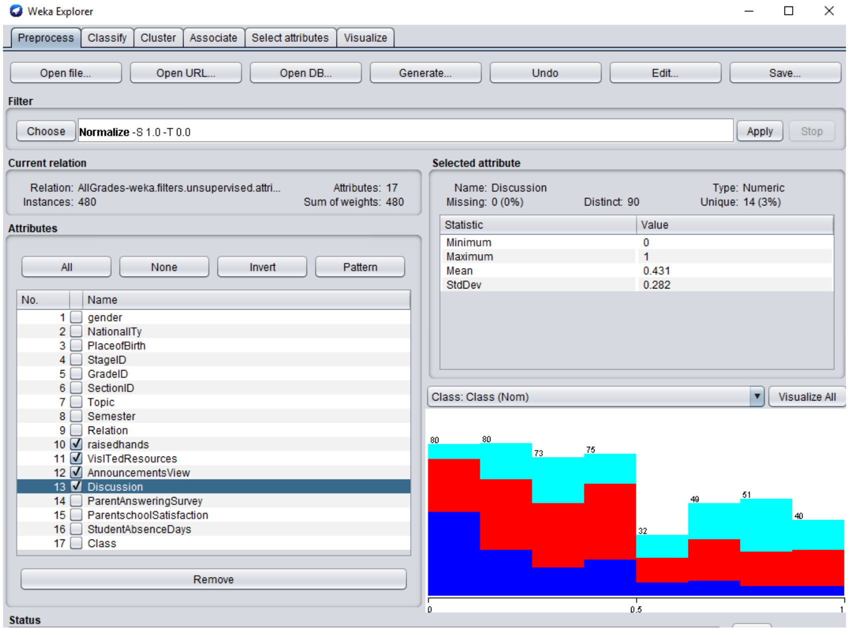


Figure 15: students.arff (After normalizing)

7. Generate Association Rules using the Apriori Algorithm

Aim: To find the association rules for the following given transactions using the Apriori algorithm and verify the same using the Weka tool and R script.

Description: The Apriori algorithm is an influential algorithm for mining frequent item sets for Boolean association rules. It uses a "bottom-up" approach, where frequent subsets are extended one at a time (a step known as candidate generation, and groups of candidates are tested against the data).

Problem: Find frequent item sets for the following transaction with a minimum support of 2 having confidence measure of 70% (i.e. 0.7)

TID	ITEMS
100	1,3,4
200	2,3,5
300	1,2,3,5
400	2,5

Solution:

Step 1: Count the number of transactions in which each item occurs

ITEM	NO. OF TRANSACTIONS
1	2
2	3
3	3
4	1
5	3

Step 2: Eliminate all those occurrences that have transaction numbers less than the minimum support (2 in this case).

ITEM	NO. OF TRANSACTIONS
1	2
2	3
3	3
5	3

These are the single items that are bought frequently. Now let's say we want to find pairs of items that are bought frequently. We continue from the above table (Table in step 2).

Step 3: We start making pairs from the first item like 1,2 1,3 1,5 and then from second item like 2,3 2,5. We do not perform 2,1 because we already did 1,2 when we were making pairs with 1 and buying 1 and 2 together is same as buying 2 and 1 together. After making all the pairs we get,

ITEM PAIRS
1,2
1,3
1,5
2,3
2,5
3,5

Step 4: Now, we count how many times each pair is bought together.

ITEM PAIRS	NO. OF TRANSACTIONS
1,2	1
1,3	2
1,5	1
2,3	2
2,5	3
3,5	2

Step 5: Again, remove all item pairs having number of transactions less than 2.

ITEM PAIRS	NO. OF TRANSACTIONS
1,3	2
2,3	2
2,5	3
3,5	2

These pair of items are bought frequently together. Now, let's say we want to find a set of three items that are bought together. We use above table (of step 5) and make a set of three items.

Step 6: To make the set of three items we need one more rule (It's termed as self-join), it simply means, from item pairs in above table, we find two pairs with the same first numeric, so, we get (2,3) and (2,5), which gives (2,3,5). Then we find how many times (2, 3, 5) are bought together in the original table and we get the following

ITEM SET	NO. OF TRANSACTIONS
(2,3,5)	2

Thus, the set of three items that are bought together from this data are (2, 3, 5).

Confidence: We can take our frequent item set knowledge even further, by finding association rules using the frequent item set. In simple words, we know (2, 3, 5) are bought together frequently, but what is the association between them. To do this, we create a list of all subsets of frequently bought items (2, 3, 5) in our case we get following subsets:

{2} {3} {5} {2,3} {3,5} {2,5}

Now, we find association among all the subsets.

{2} => {3,5}: If '2' is bought, what's the probability that '3' and '5' would be bought in same transaction, confidence = P (3ᴗ5ᴗ2)/ P(2) =2/3 =67%
{3}=>{2,5}= P (3ᴗ5ᴗ2)/ P(3)=2/3=67%
{5}=>{2,3}= P (3ᴗ5ᴗ2)/ P(5)=2/3=67%
{2,3}=>{5}= P (3ᴗ5ᴗ2)/ P(2ᴗ3)=2/2=100%
{3,5}=>{2}= P (3ᴗ5ᴗ2)/ P(3ᴗ5)=2/2=100%
{2,5}=>{3}= P (3ᴗ5ᴗ2)/ P(2ᴗ5)=2/3=67%

Also, considering the remaining 2-items sets, we would get the following associations:
{1}=>{3}=P(1ᴗ3)/P(1)=2/2=100%
{3}=>{1}=P(1ᴗ3)/P(3)=2/3=67%
{2}=>{3}=P(3ᴗ2)/P(2)=2/3=67%
{3}=>{2}=P(3ᴗ2)/P(3)=2/3=67%
{2}=>{5}=P(2ᴗ5)/P(2)=3/3=100%
{5}=>{2}=P(2ᴗ5)/P(5)=3/3=100%
{3}=>{5}=P(3ᴗ5)/P(3)=2/3=67%
{5}=>{3}=P(3ᴗ5)/P(5)=2?3=67%

Eliminate all those having confidence less than 70%. Hence, the rules would be:
{2,3}=>{5}, {3,5}=>{2}, {1}=>{3},{2}=>{5}, {5}=>{2}
Now these manual results should be checked with the rules generated in Weka. So first create a CSV file for the above problem; the CSV file for the above problem will look like the rows and columns in the following figure:

	A	B	C	D	E	F
1	I1	I2	I3	I4	I5	
2	t		t	t		
3		t	t		t	
4	t	t	t		t	
5		t			t	
6						
7						
8						
9						

Figure 16: Input data (CSV)

Procedure for generating the rules in Weka:

Step 1: Open the Weka explorer and open the input CSV file that we have created. Then select all the item sets.

Step 2: Now select the association tab and then choose Apriori algorithm by setting the minimum support and confidence as shown in the figure.

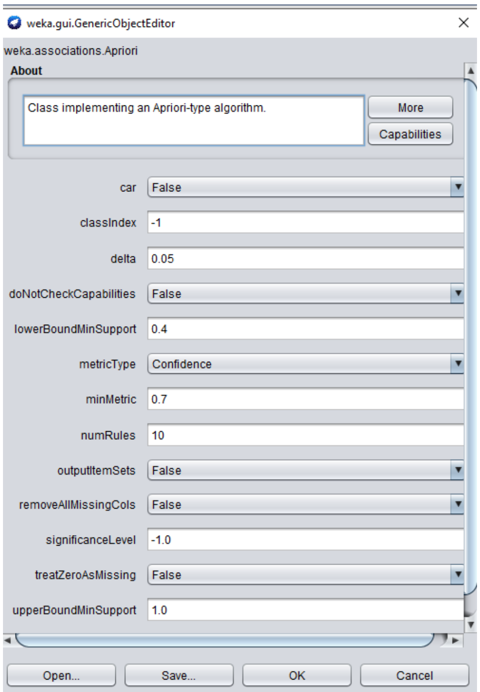


Figure 17: Apriori parameters

Step 3: Now click on 'Start'.

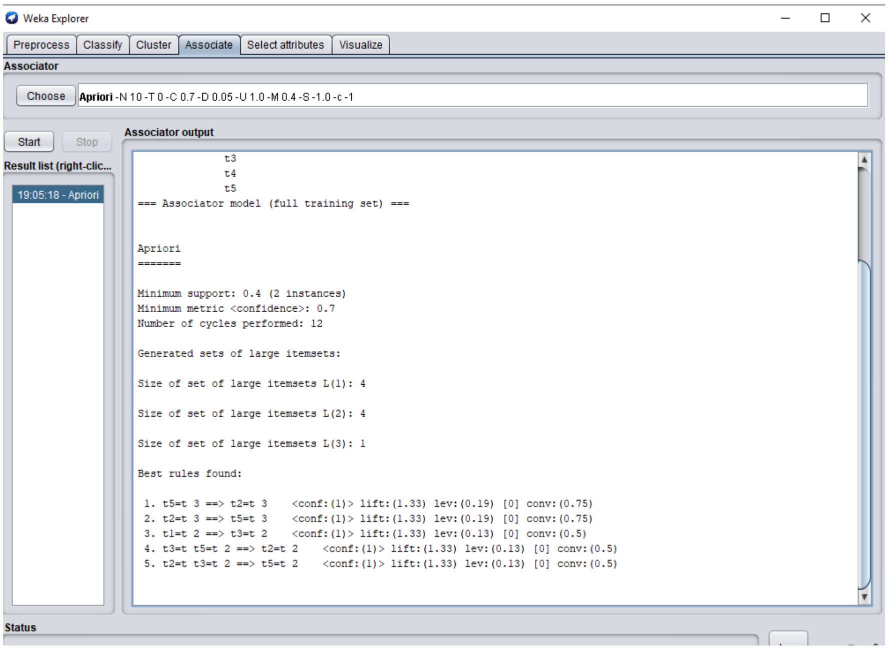


Figure 18: Association rules generated using Weka

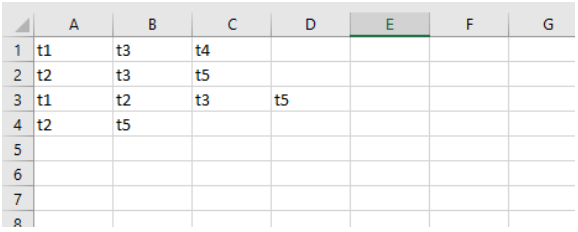
Conclusion: As we have seen the total rules generated by us manually and by the Weka are matching.

R Script for Association analysis:

#includes the libraries

```
library(arules)
#finding association rules
txn <- read.transactions(file="G:/Academic/4/4-1/DM Lab/apriori/input.csv", rm.duplicates= TRUE,
format="basket",sep=",",cols=NULL)
txn@itemInfo$labels <- gsub("\"","",txn@itemInfo$labels)
basket_rules <- apriori(txn,parameter = list(sup = 0.4, conf = 0.7,target="rules"));
inspect(basket_rules)
#Plot the baskets
itemFrequencyPlot(txn, topN = 5)
```

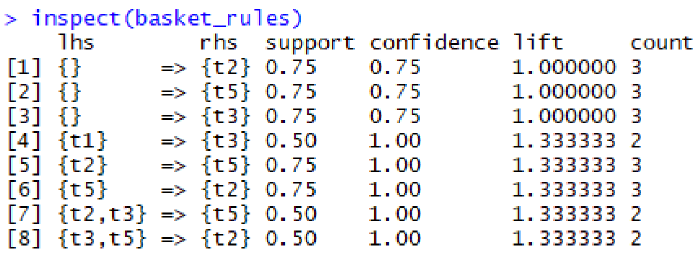
Input (groceries dataset):



	A	B	C	D	E	F	G
1	t1	t3	t4				
2	t2	t3	t5				
3	t1	t2	t3	t5			
4	t2	t5					
5							
6							
7							

Figure 19: Input data (CSV)

Output:



```
> inspect(basket_rules)
    lhs        rhs  support confidence lift     count
[1] {}      => {t2} 0.75    0.75       1.000000 3
[2] {}      => {t5} 0.75    0.75       1.000000 3
[3] {}      => {t3} 0.75    0.75       1.000000 3
[4] {t1}    => {t3} 0.50    1.00       1.333333 2
[5] {t2}    => {t5} 0.75    1.00       1.333333 3
[6] {t5}    => {t2} 0.75    1.00       1.333333 3
[7] {t2,t3} => {t5} 0.50    1.00       1.333333 2
[8] {t3,t5} => {t2} 0.50    1.00       1.333333 2
```

Figure 20: Output

Graph:

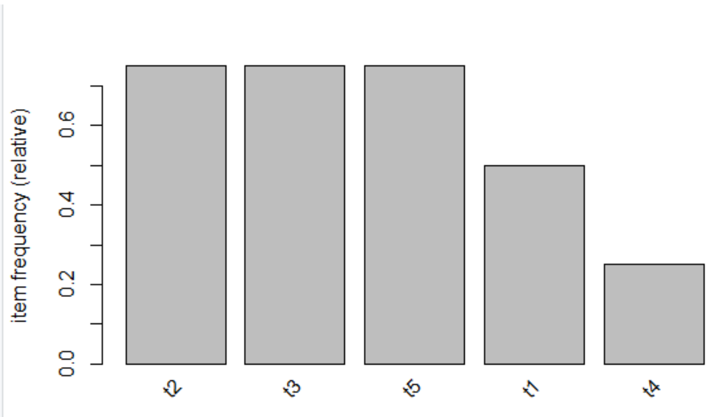


Figure 21: Items count

8. Generating Association Rules Using FP Growth Algorithm

Aim: To generate association rules using the FP Growth Algorithm and verify it using Weka and R scripting.

Description: The FP-Growth Algorithm, proposed by Han is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm and the TreeProjection. In some later works it was proved that FP-Growth has better performance than other methods, including Eclat and Relim. The popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve his performance.

Problem: To find all frequent item sets in following dataset using FP-growth algorithm. Minimum support=2 and confidence =70%

TID	ITEMS
100	1,3,4
200	2,3,5
300	1,2,3,5
400	2,5

Solution: Similar to Apriori Algorithm, find the frequency of occurrences of all each item in dataset and then prioritize the items according to its descending order of its frequency of occurrence. Eliminating those occurrences with the value less than minimum support and assigning the priorities, we obtain the following table.

ITEM	NO. OF TRANSACTIONS	PRIORITY
1	2	4
2	3	1
3	3	2
5	3	3

Re-arranging the original table, we obtain-

TID	ITEMS
100	3,1
200	2,3,5
300	2,3,5,1
400	2,5

Construction of tree: Note that all FP trees have ‘null’ node as the root node. So, draw the root node first and attach the items of the row 1 one by one respectively and write their occurrences in front of it. The tree is further expanded by adding nodes according to the prefixes (count) formed and by further incrementing the occurrences every time they occur and hence the tree is built.

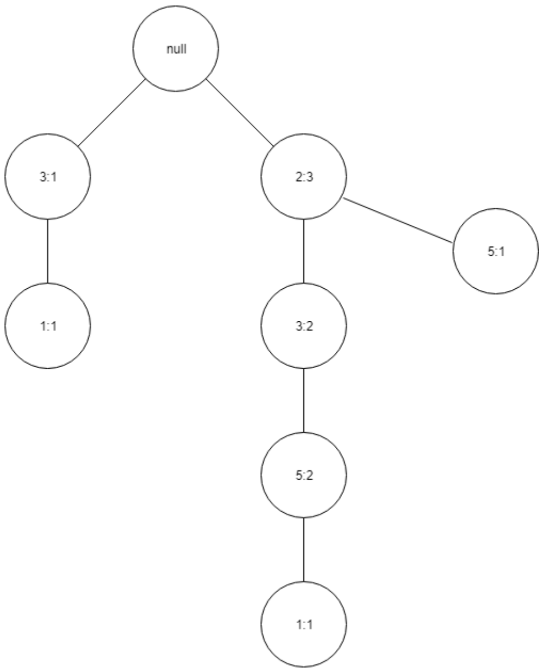


Figure 22: FP Tree

Prefixes:

1 -> 3:1
1 -> 2,3,5:1
5 -> 2,3:2
5 -> 2:1
3 -> 2:2

Frequent item sets:

1-> 3:2
3 -> 2 :2
5 -> 2:3
5 -> 3:2
5 -> 2,3:2

Therefore, the frequent item sets are {1,3}, {2,3}, {2,5}, {3,5}, {2,3,5}

Generating the association rules for the following tree and calculating the confidence measures we get:

{3}=>{1}=2/3=67%
{1}=>{3}=2/2=100%
{2}=>{3,5}=2/3=67%
{2,5}=>{3}=2/3=67%
{3,5}=>{2}=2/2=100%
{2,3}=>{5}=2/2=100%
{3}=>{2,5}=2/3=67%
{5}=>{2,3}=2/3=67%
{2}=>{5}=3/3=100%
{5}=>{2}=3/3=100%
{2}=>{3}=2/3=67%
{3}=>{2}=2/3=67%

Thus, eliminating all the sets having confidence less than 70%, we obtain the following conclusions:

{1} => {3}, {3,5} => {2}, {2,3} => {5}, {2}=>{5}, {5} => {2}

As we see there are 5 rules that are being generated manually and these are to be checked against the results in WEKA. In order to check the results in the tool we need to follow the similar procedure like Apriori.
So first create a CSV file for the above problem, the CSV file for the above problem will look like the rows and columns in the above figure. This file is written in an Excel sheet.

	A	B	C	D	E
1	I1	I2	I3	I4	I5
2	t		t	t	
3		t	t		t
4	t	t	t		t
5		t			t
6					
7					

Figure 23: Input data (CSV)

Procedure for running the rules in Weka:

Step 1: Open Weka Explorer and open the file and then select all the item sets.

Step 2: Now select the Associate tab and then choose FP Growth algorithm by setting the minimum support and confidence as shown in the figure.

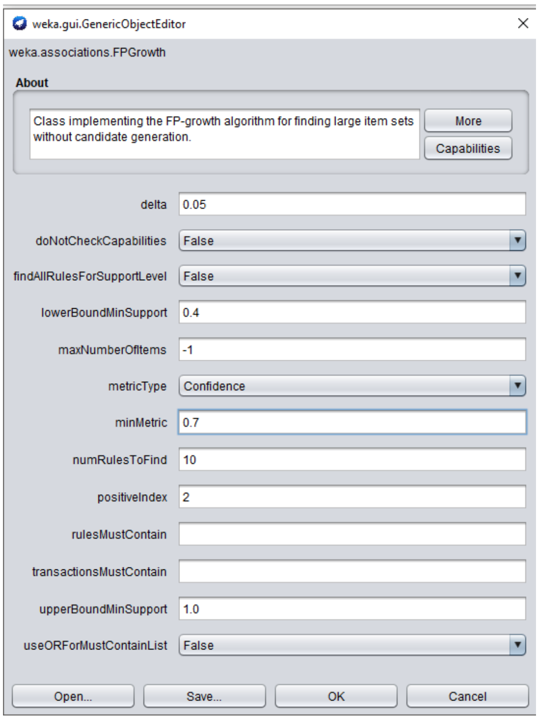


Figure 24: FP Growth input parameters

Step 3: Now run the FP Growth algorithm with the set values of minimum support and the confidence. After running the Weka generates the association rules and the respective confidence with minimum support as shown in the figure.

The above CSV file has generated 5 rules as shown in the figure:

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.7 -D 0.05 -U 1.0 -M 0.4
Relation:     data
Instances:    4
Attributes:   5
              t1
              t2
              t3
              t4
              t5
=== Associator model (full training set) ===

FPGrowth found 5 rules (displaying top 5)

1. [t5=t]: 3 ==> [t2=t]: 3   <conf:(1)> lift:(1.33) lev:(0.19) conv:(0.75)
2. [t2=t]: 3 ==> [t5=t]: 3   <conf:(1)> lift:(1.33) lev:(0.19) conv:(0.75)
3. [t1=t]: 2 ==> [t3=t]: 2   <conf:(1)> lift:(1.33) lev:(0.13) conv:(0.5)
4. [t5=t, t3=t]: 2 ==> [t2=t]: 2   <conf:(1)> lift:(1.33) lev:(0.13) conv:(0.5)
5. [t3=t, t2=t]: 2 ==> [t5=t]: 2   <conf:(1)> lift:(1.33) lev:(0.13) conv:(0.5)
```

Figure 25: Output of FP Growth (Weka)

Conclusion: We have thus seen the total rules generated by us manually and by Weka are matching.

9. Build a Decision Tree by using the J48 algorithm

Aim: Generate a Decision Tree by using J48 algorithm.

Description: Decision tree learning is one of the most widely used and practical methods for inductive inference over supervised data. It represents a procedure for classifying categorical databased on their attributes. This representation of acquired knowledge in tree form is intuitive and easy to assimilate by humans.

The entropy is a measure of the uncertainty associated with a random variable. As uncertainty increases, so does entropy, values range from [0-1] to present the entropy of information

$$\text{Entropy (D)} = \sum_{j=1}^{c} -plog_2p$$

Information gain is used as an attribute selection measure; pick the attribute having the highest information gain, the gain is calculated by:

$$\text{Gain (D, A)} = \text{Entropy (D)} - \sum_{j=1}^{v} \frac{|Dj|}{|D|} \text{Entropy(Dj)}$$

where, D: A given data partition

A: Attribute

V: Suppose we were partition the tuples in D on some attribute A having v distinct values D is split into v partition or subsets, (D1, D2… Dj), where Dj contains those tuples in D that have outcome Aj of A.

Problem: Build a decision tree for the following data

AGE	INCOME	STUDENT	CREDIT_RATING	BUYS_COMPUTER
Youth	High	No	Fair	No
Youth	High	No	Excellent	No
Middle aged	High	No	Fair	Yes
Senior	Medium	No	Fair	Yes
Senior	Low	Yes	Fair	Yes
Senior	Low	Yes	Excellent	No
Middle aged	Medium	Yes	Excellent	Yes
Youth	Low	No	Fair	No
Youth	Medium	Yes	Fair	Yes
Senior	Medium	Yes	Fair	Yes
Youth	Medium	Yes	Excellent	Yes
Middle aged	Medium	No	Excellent	Yes
Middle aged	High	Yes	Fair	Yes
Senior	Medium	No	Excellent	No

Class P: buys_computer="yes"
Class N: buys_computer="no"
Entropy (D) = -9/14log (9/14)-5/15log (5/14) =0.940

Compute the expected information requirement for each attribute start with the attribute age

Gain (age, D) = Entropy(D) - $\sum_{youth,middle-aged,senior}^{n} \left(\frac{Sv}{S}\right) Entropy(Sv)$

= Entropy(D) - $\frac{5}{14}$Entropy(S_{youth}) - $\frac{4}{14}$Entropy($S_{middle\text{-}aged}$) - $\frac{5}{14}$Entropy(S_{senior})
= 0.940-0.694
= 0.246

Similarly, for other attributes
Gain (income, D) =0.029
Gain (student, D) = 0.151
Gain (credit_rating, D) = 0.04

The attribute age has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of age. These tuples are shown partitioned accordingly.

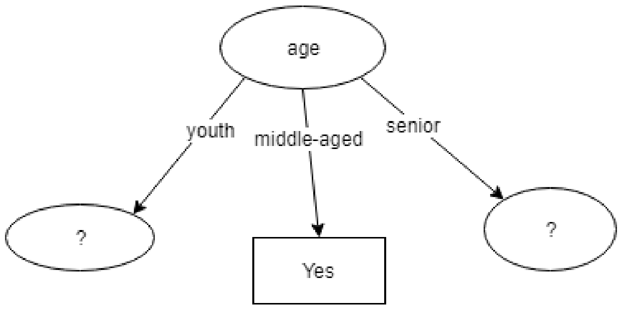


Figure 26: Decision Tree 1

AGE	INCOME	STUDENT	CREDIT_RATING	BUYS_COMPUTER
Youth	High	No	Fair	No
Youth	High	No	Excellent	No
Youth	Low	No	Fair	No
Youth	Medium	Yes	Fair	Yes
Youth	Medium	Yes	Excellent	Yes

Now, calculating information gain for sub-table (age<=30)
I (2,3) = -(2/5)log(2/5)-(3/5)log(3/5)=0.971
INCOME
Income="high" S11=0, S12=2
I=0
Income="medium" S21=1 S22=1
I (S21, S23) = 1

Income=”low” S31=1 S32=0

I=0

Entropy for income

E(income) = (2/5)(0) + (2/5)(1) + (1/5)(0) = 0.4

Gain(income) = 0.971 - 0.4 = 0.571

Similarly,

Gain(student)=0.971

Gain(credit)=0.0208

Gain(student) is highest

AGE	INCOME	STUDENT	CREDIT_RATING	BUYS_COMPUTER
Senior	Medium	No	Fair	Yes
Senior	Low	Yes	Fair	Yes
Senior	Low	Yes	Excellent	No
Senior	Medium	Yes	Fair	Yes
Senior	Medium	No	Excellent	No

For a Senior,

information gain= -(3/5)log(3/5)-(2/5)log(2/5)=0.971

income

income=”low” I=1,

income=”medium” I=0.917

E(income)=(2/5)(1)+(3/5)(0.917)=0.9502

Gain(income)=0.0208

Similarly,

gain(student)=0.0208,

gain(credit)=0.971 and this would result in final decision tree:

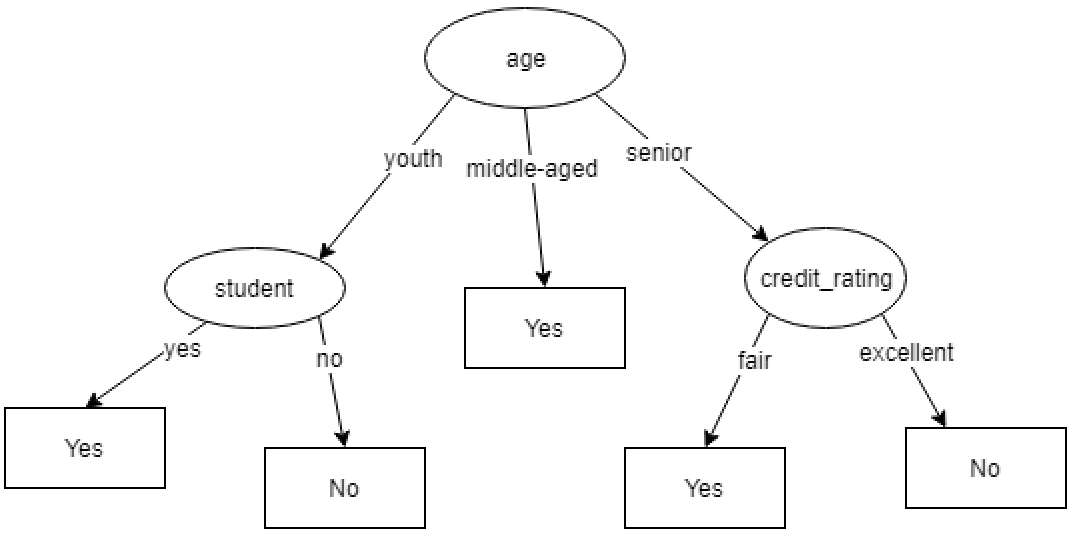
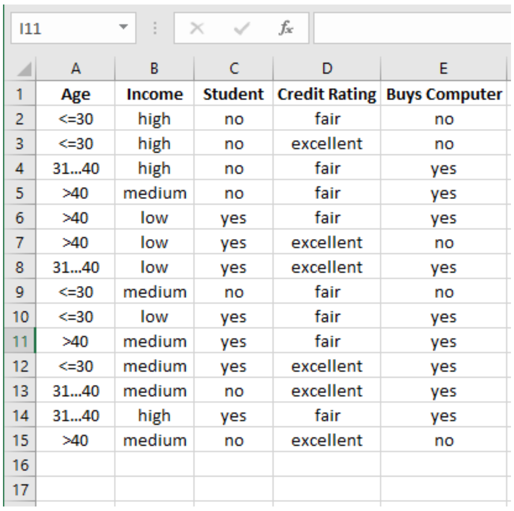


Figure 27: Decision Tree (complete)

Procedure for running the rules in Weka:

First create a CSV file for the above problem; the csv file for the above problem will look like the rows and columns in the below figure. This file is written in an Excel sheet.



	A	B	C	D	E
1	Age	Income	Student	Credit Rating	Buys Computer
2	<=30	high	no	fair	no
3	<=30	high	no	excellent	no
4	31...40	high	no	fair	yes
5	>40	medium	no	fair	yes
6	>40	low	yes	fair	yes
7	>40	low	yes	excellent	no
8	31...40	low	yes	excellent	yes
9	<=30	medium	no	fair	no
10	<=30	low	yes	fair	yes
11	>40	medium	yes	fair	yes
12	<=30	medium	yes	excellent	yes
13	31...40	medium	no	excellent	yes
14	31...40	high	yes	fair	yes
15	>40	medium	no	excellent	no
16					
17					

Figure 28: Input data (Weka)

Procedure for running the rules in Weka:

Step 1: Open Weka explorer and open the file and then select all the item sets.

Step 2: Now select the classify tab in the tool and click on start button.

Step 3: Check the main result which we got manually and the result in Weka by right clicking on the result and visualizing the tree.

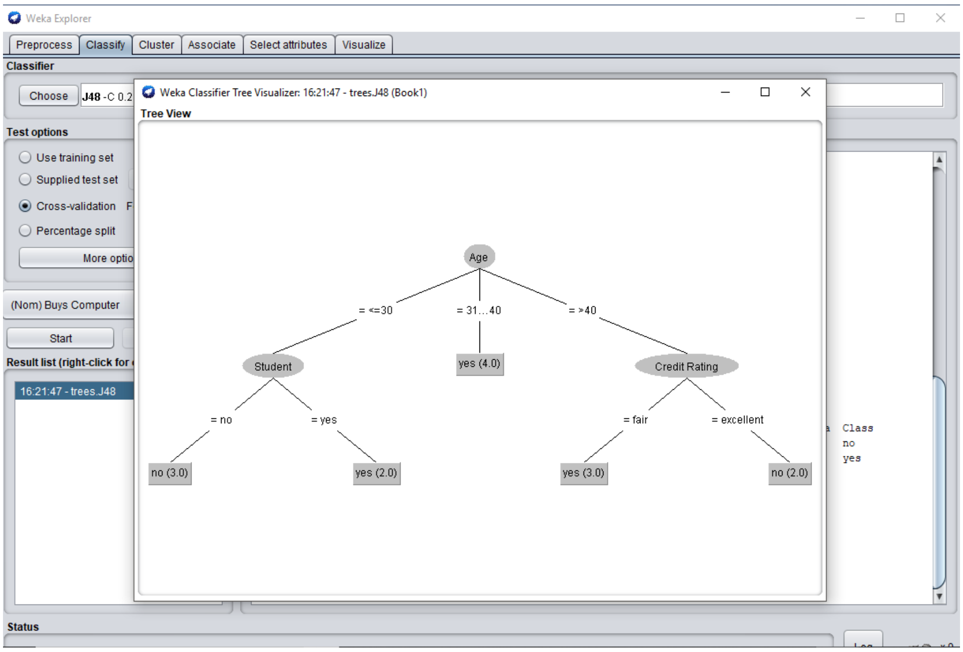


Figure 29: Weka Output

R script for Decision Tree:

```
#load libraries
library('rpart')
library('rpart.plot')
# Classification Tree with rpart
library(rpart)
input.dat = read.csv("C:/Users/HP/Desktop/Book1.csv",header=TRUE,sep =",")
input.dat
# grow tree
fit <- rpart(buys~age+income+student+credit_rating, method='class',data=input.dat, control
=rpart.control(minsplit = 1,minbucket=1, cp=0))
# plot tree
plot(fit, uniform=TRUE, main="Tree")
text(fit, use.n=TRUE, all=TRUE, cex=.8)
```

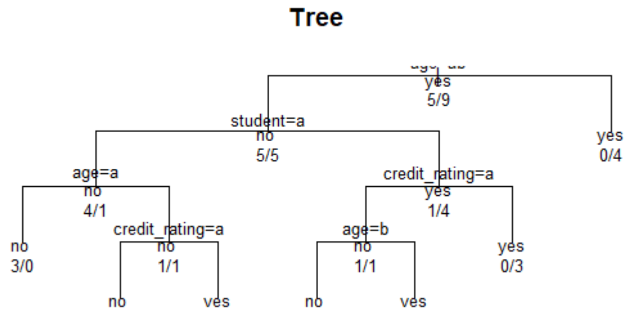


Figure 30: RStudio Output

10. Naïve Bayes Classification on a given data set

Aim: To apply Naïve Bayes classifier on a given data set.

Description: In machine learning, Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' Theorem with strong (naïve) independence assumptions between the features.

Problem:

AGE	INCOME	STUDENT	CREDIT_RATING	BUYS_COMPUTER
<=30	High	No	Fair	No
<=30	High	No	Excellent	No
31..40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31..40	Medium	Yes	Excellent	Yes
<=30	Low	No	Fair	No
<=30	Medium	Yes	Fair	Yes
>40	Medium	Yes	Fair	Yes
<=30	Medium	Yes	Excellent	Yes
31..40	Medium	No	Excellent	Yes
31..40	High	Yes	Fair	Yes
>40	Medium	No	Excellent	No

Classes:

C1: buys_computer = 'yes'
C2: buys_computer='no'

Data to be classified:

X= (age<=30, income=Medium, Student=Yes, credit_rating=Fair)

- P(buys_computer="yes")= 9/14 = 0.643
  P (buys_computer="no") =5/14=0.357

- Compute P(X/C1) and P(X/C2), we get:

  P(age=”<=30” | buys_computer=”yes”)=2/9
  P(age=”<=30”| buys_computer=”no”)=3/5
  P(income=”medium ”| buys_computer=”yes”)=4/9
  P(income=”medium ”| buys_computer=”no”)=2/5
  P(student=”yes ”| buys_computer=”yes”)=6/9
  P(student=”yes” | buys_computer=”no”)=1/5=0.2
  P(credit_rating=”fair ”| buys_computer=”yes”)=6/9
  P(credit_rating=”fair” | buys_computer=”no”)=2/5

- X=(age<=30, income=medium, student=yes, credit_rating=fair)

  P(X|C1)=2/9*4/9*6/9*6/9=32/1134
  P(X|C2)=3/5*2/5*1/5*2/5=12/125
  P(C1|X) = P(X|C1)P(C1)
  =(32/1134)*(9/14)
  =0.019
  P(C2/X) = p(X/C2)*p(C2)
  = (12/125)*(5/14)
  = 0.007

  Therefore, conclusion is that the given data belongs to C1 since P(C1/X)>P(C2/X)

Checking the result on Weka:

In order to check the result in the tool we need to follow a procedure:

Step 1: Create a CSV file with the above table considered in the example.

Step 2: Now open Weka explorer and then select all the attributes in the table.

Step 3: Select the classifier tab in the tool and choose Bayes folder and then Naïve Baye’s classifier to see the result as shown below.

```
=== Summary ===

Correctly Classified Instances           0                0      %
Incorrectly Classified Instances         1              100      %
Kappa statistic                          0
Mean absolute error                      0.7538
Root mean squared error                  0.7538
Relative absolute error                120.6124 %
Root relative squared error            120.6124 %
Total Number of Instances                1

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    1.000    0.000      0.000    0.000      0.000    ?         ?         yes
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         1.000     no
Weighted Avg.    0.000    0.000    0.000      0.000    0.000      0.000    0.000     1.000

=== Confusion Matrix ===

 a b   <-- classified as
 0 0 | a = yes
 1 0 | b = no
```

Figure 31: Weka Output

R program for Naïve Bayes Classifier:

```
library(e1071)
library(rjson)
#Load Data
input.dat = read.csv("C:/Users/HP/Desktop/Book1.csv",header=TRUE,sep =",")
#Train the Model
model <- naiveBayes(buys~age+income+student+credit_rating, data = input.dat)
#Predict using that model
predict(model, input.dat[1,-5])
predict(model, input.dat[3,-5])
```

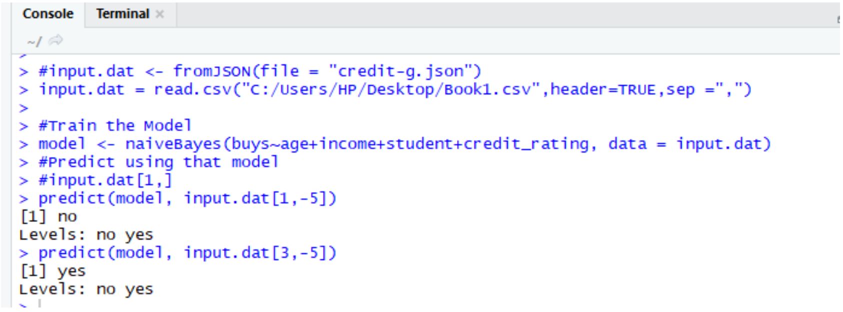


Figure 32: RStudio Output

11. K-means Clustering

Aim: Using K-means to perform clustering and verify the result using Weka and R scripting.

Description: K-means algorithm aims to partition n observations into "k clusters" in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Problem: As a simple illustration of a k-means algorithm, consider the following data set consisting of the scores of two variables on each of the five variables.

I	**X**	**Y**
A	1	1
B	1	0
C	0	2
D	2	4
E	3	5

This data set is to be grouped into two clusters: As a first step in finding a sensible partition, let the A & C values of the two individuals furthest apart (using the Euclidean distance measure), define the initial cluster means, giving:

Cluster	Individual	Mean Vector(Centroid)
Cluster 1	A	(1,1)
Cluster 2	C	(0,2)

The remaining individuals are now examined in sequence and allocated to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean. The mean vector is recalculated each time a new member is added. This leads to the following series of steps:

	A	C
A	0	1.4
B	1	2.5
C	1.4	0
D	3.2	2.82
E	4.5	4.2

Initial partitions have changed, and the two clusters at this stage having the following characteristics.

	Individual	Mean vector(Centroid)
Cluster 1	A,B	(1,0.5)
Cluster 2	C,D,E	(1.7,3.7)

But we cannot yet be sure that each individual has been assigned to the right cluster. So, we compare each individual’s distance to its own cluster mean and to that of the opposite cluster. And, we find:

I	A	C
A	0.5	2.7
B	0.5	3.7
C	1.8	2.4
D	3.6	0.5
E	4.9	1.9

The individuals C is now relocated to Cluster 1 due to its less mean distance with the centroid points. Thus, it’s relocated to cluster 1 resulting in the new partition

	Individual	Mean vector(Centroid)
Cluster 1	A,B,C	(0.7,1)
Cluster 2	D,E	(2.5,4.5)

The iterative relocation would now continue from this new partition until no more relocation occurs. However, in this example each individual is now nearer its own cluster mean than that of the other cluster and the iteration stops, choosing the latest partitioning as the final cluster solution.

Also, it is possible that the k-means algorithm won’t find a final solution. In this case, it would be a better idea to consider stopping the algorithm after a pre-chosen maximum number of iterations.

Verifying using Weka:

In order to check the result in the tool we need to follow a procedure.

Step 1: Create a csv file with the above table considered in the example. The CSV file will look as shown below:

	A	B	C	D
1	i	x1	x2	
2	A	1	1	
3	B	1	0	
4	C	0	2	
5	D	2	4	
6	E	3	5	
7				

Figure 33: Input Weka Data

Step 2: Now open Weka explorer and then select all the attributes in the table.

Step 3: Select the cluster tab in the tool and choose normal k-means technique to see the result as shown below.

```
Initial starting points (random):

Cluster 0: D,2,4
Cluster 1: B,1,0

Missing values globally replaced with mean/mode

Final cluster centroids:
                         Cluster#
Attribute    Full Data          0          1
                 (5.0)      (2.0)      (3.0)
============================================
ï»¿i                 A          D          A
x1                 1.4        2.5     0.6667
x2                 2.4        4.5          1



Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       2 ( 40%)
1       3 ( 60%)
```

Figure 34: Weka Output

Rscript for K-means:

```
#load the required packages
#ggplot is used to create plots
library(ggplot2)
#plot the iris data set
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
#clustering
set.seed(20)
#forming a kmeans cluster
irisCluster <- kmeans(iris[,3:4], 3, nstart=20)
```

```
irisCluster
#plot to see the clusters
irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Is.Versicolor	Predict.Versicolor.lm	Predict.Versicolor.logit
1	5.1	3.5	1.4	0.2	setosa	0	0	0
2	4.9	3.0	1.4	0.2	setosa	0	0	0
3	4.7	3.2	1.3	0.2	setosa	0	0	0
4	4.6	3.1	1.5	0.2	setosa	0	0	0
5	5.0	3.6	1.4	0.2	setosa	0	0	0
6	5.4	3.9	1.7	0.4	setosa	0	0	0
7	4.6	3.4	1.4	0.3	setosa	0	0	0
8	5.0	3.4	1.5	0.2	setosa	0	0	0
9	4.4	2.9	1.4	0.2	setosa	0	0	0
10	4.9	3.1	1.5	0.1	setosa	0	0	0

Figure 35: Iris dataset (input)

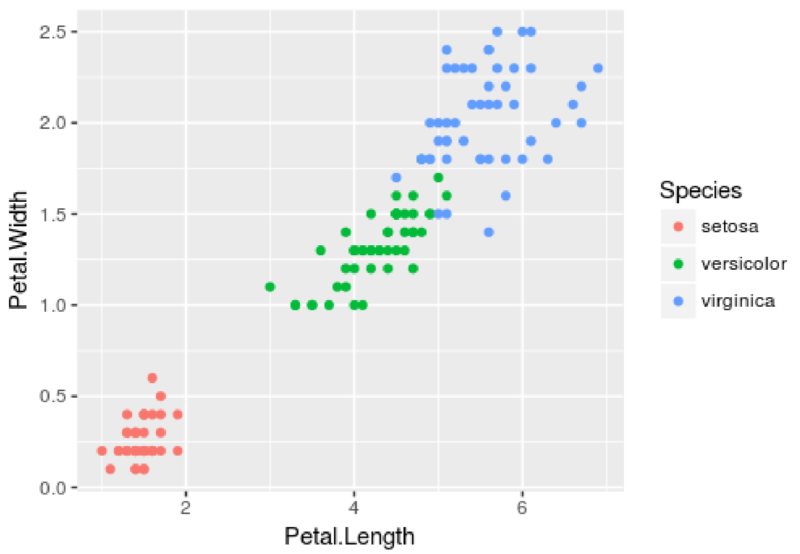


Figure 36: RStudio Graphical Output

12. Regression analysis

Aim: To generate a linear regression using R

Description: Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.
The general mathematical equation for a linear regression is −

$y = ax + b$

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

R Script to perform regression analysis:

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
# Give the chart file a name.
png(file = "linearregression.png")
plot(y,x,col = "blue",main = "Height & Weight Regression", abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight
in Kg",ylab = "Height in cm")
```

```
# Save the file.
dev.off()
```

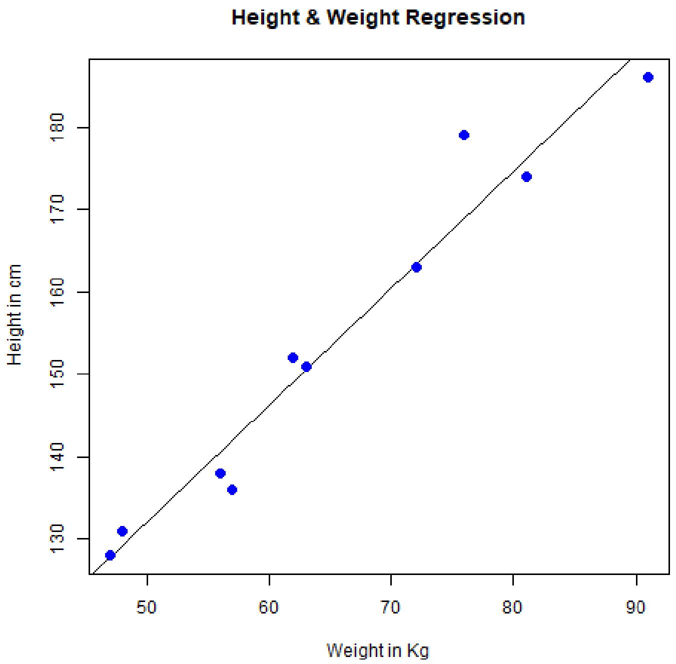


Figure 37: RStudio Output