(7082CEM)
Coursework
Demonstration of a Big Data Program

MODULE LEADER: Dr. Marwan Fuad

Student Name: Rohith Chityala

SID: 10013258

ANALYSIS OF BIG DATA USING PYSPARK AND MACHINE LEARNING

I can confirm that all work submitted is my own: Yes

Table of Contents

1. Introduction

In recent years, Big Data has been of the hot topics being discussed around the world as its implementations are expanding into numerous industries rapidly. To put it in lament terms, Big Data is a description of large amounts of structures, semi-structured and unstructured data that can be analysed. Astonishingly, 2.4 quintillion bytes of data is being generated every day in the year 2021. According to a survey, ninety per cent of the data available today in the world has been created in the last two years.
Generally, Big Data is defined by five different parameters known as five V's of Big Data. These are volume, velocity, variety, veracity, value. Volume is known as the base of the Big Data as it defines the amount of data being generated. Velocity determines how quick the data flowing happens. It is important to note that it's beneficial to have limited data at a fast speed than data generating at a slow pace. Variety describes how different is the data available and Veracity denotes how clean and real is the data available. Finally, value of the data gives us an idea about how much useful are the insights drawn from analysing the data.
In this study, I am going to analyse Who Eats the Food We Grow dataset using PySpark and Machine Learning. The dataset is taken from the Kaggle website https://www.kaggle.com/dorbicycle/world-foodfeed-production. Food production and its supplying all over the world has always been a challenge to the humankind. Day to day increase in population of the world is forcing humans to invent unordinary ways of producing food and transporting it to various parts of the world. Not only agricultural production, but also quality maintenance of the food produced is becoming critical for the mankind to survive on earth. As per the U.N. the world population is going to hit 9.3 billion by the year 2050, so many countries are investing huge amounts of money in food research and advancement of agriculture development technologies. According to the Food and Agriculture Organization (FAO), there should be an overall 70% increase in the food production to feed the population by 2050.
On the other hand, there's also an issue of producing adequate amounts of feed for the animals to maintain balance of the food chain. The challenges carried by the feed production are also huge and they will only grow as the population increases. Hence it is very important for the countries to establish futuristic technologies and methods of producing food and feed.
Food and Agricultural Organization data gives us the explanation of both food and feed generated in the world from 1961 to 2013. Analysing this data and gaining insights from it would help countries and organizations in understanding, researching, and developing new techniques for producing agriculture.

2. Related Work

2.1 Apache Spark

Spark is a popular, largest open-source platform available for data processing. Its applications are mainly found in Big Data, Machine Learning, Interactive Analysis, and Fog Computing. Many notable companies such as Netflix, Uber, Pinterest etc are using Spark intensively for data analysing which further improves the customer experience.

One more special feature Spark offers is high-level APIs available in different languages such as Python, Java, and Scala.

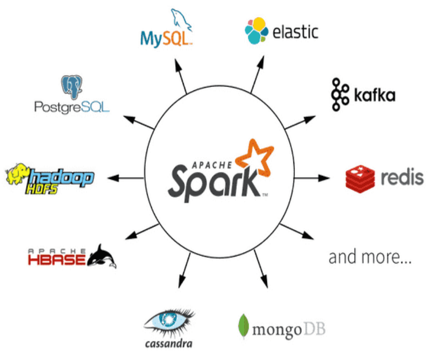


Fig 1.1. Spark Databases

Spark can also be used by connecting to different databases such as MySQL, Hive, Cassandra etc. as shown in the fig 1.1.

2.1.1 Features of Spark

Apache Spark has various special features which it is famous for. The features that make Spark the most preferred platform for processing and analysing data are as follows.

- Speed
- Usability
- Advanced analytics
- Runs everywhere
- In-memory computing
- Real-time stream processing

2.1.2 Spark Ecosystem

Various components and projects are being developed in Spark and some of the important components are mentioned below.

1. **Spark Core:**
   Spark Core can be called as the heart of the Spark platform. The execution engine and processing unit is provided by Spark Core which then is used by other components. It also helps in accessing data from external storage and provides in-built memory processing. Its main duty is to monitor, perform, and process all I/O functions. Apart from these, Spark has some important functions such as fault tolerance, memory management.

2. **Spark SQL:**
   The name Spark SQL is obtained because it works and processes data in the way like SQL. Before Spark SQL, Shark was there to handle this type of tasks, but due to its ineffectiveness it was replaced by Spark SQL. Its main aim is to allow programmers to not worry about the way data is distributed and focus on business problems. It is used to manage, locate, handle data from internal and external memory.

3. **Spark Streaming:**
   Spark Streaming is one of the vital libraries of Spark. Its primary function is to allow the Spark operations to be done at a fast speed. In fact, it can process gigabytes of data per second. But how does it do this? The question to this answer is very simple as it need not

process all the data at once, instead it makes the data into different chunks and process them. This makes Spark a very reliable source for large amounts of data.

4. **MLlib:**
   Spark also offers an amazing library called MLlib which allows users to perform all the machine learning algorithms in Spark. It can be called using Java, Python and Scala. It is very easy and simple to use. Today, companies rely on day-to-day data of customer experience and learn from the outputs drawn from it. This whole thing is done by the machine learning algorithms, so Spark also included this library in its system.

5. **GraphX:**
   Graph in Apache Spark doesn't mean the general graphical representations. This API offered by Spark is used for graphs and graph-parallel computation. Basically, it is a concept of defining relationships between nodes and edges, where nodes are the vertices or units, and edges are called the relationships formed between these nodes.

2.1.3 SparkSession

SparkSession is the entry point for any function to be implemented in spark. Prior to version spark version 2, SparkContext was used as an entry point. Now, SparkSession internally consists of both SparkContext and SparkConfig. To build a SparkSession, builder() function is used and to for calling it, getorcreate() function is used. So SparkSession would be the first statement in our program to create or load a dataset through dataframe or RDD. Some of the functions that can be used are createDataFrame(), emptyDataFrame(), createDataset(), getActiveSession() etc.

2.1.4 DataFrame:

A DataFrame is defined as the collection of distributed data which is grouped into named columns. A dataframe can be constructed from multiple form of external databases such as Hive or SQL. The dataframe is known for its ability to process kilobytes to petabytes of data on a single node cluster and it accepts different forms of data formats such as .csv,. json, Cassandra etc.

2.1.5 Resilient Distributed Dataset (RDD)

The RDD is the first and foremost data structure in Spark. It is a collection of distributed objects which is immutable that means it cannot be changed. So RDD is a read only collection of records. Basically, RDD can be created using two ways- one is with parallelising the already available data
and the other one is by accessing the data from an external database or source. RDD helps spark in making fast and most efficient MapReduce operations.

2.2 PySpark:

As we know spark has many different applications within its own and combined with other components. One such amazing feature offered by spark is PySpark which is a combination of python and spark. Python language implemented in spark is known as PySpark.
It has some of the widely utilized machine learning techniques such as tensorflow, pandas and numpy so its applications are vast in the machine learning and data science areas. PySpark is a great language for performing exploratory data analysis at scale, building machine learning pipelines, and creating ETLs for a data platform. If you're already familiar with Python and libraries such as Pandas, then PySpark is a great language to learn to create more scalable analyses and pipelines.

2.3 The Dataset:

The FAO dataset has been taken from the Kaggle website https://www.kaggle.com/dorbicycle/world-foodfeed-production. The Food and Agricultural Organization provides free access to the food production data from year 1961 to the recent year. A part of this dataset has been taken for analysis in our report. The dataset used contains data of food and feed produced all over the world from 1961-

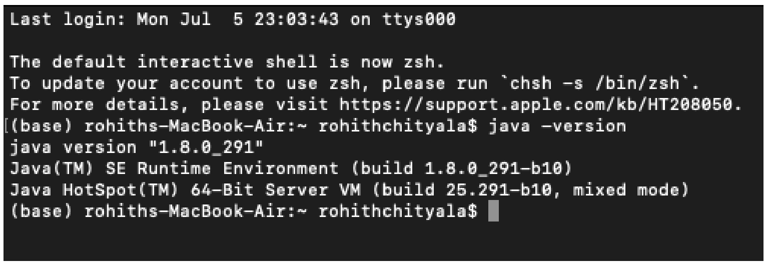
2013. There is total 63 attributes/columns of which consists of multiple datatypes. The column element consists of two types of production- food and feed. Food denotes the substances produced for humans and feed is the production of food for animals. The below table gives us clear idea about the columns present in the dataset.

Features	Definition
Area Abbreviation	Area abbreviation is the shortcut word for Area.
Area Code	Area code consists of codes for all the countries present.
Area	Area column consists of list of countries taken for the study.
Item Code	It gives us the codes for each different item produced.
Item	The types of food and feed items produced by different countries.
Element Code	Gives us the codes for food and feed.
Element	There are two elements present in the dataset- Food and Feed.
Unit	The amount of food and feed produced by each country per year.
Latitude	It gives us the latitude information about the Area.
Longitude	It gives us the longitude information about the Area.
Y1961-Y2013(all other columns)	These columns contain the amount of food items produced per 1000 tonnes.

2.4 Implementation

2.4.1 Installation and PySpark setup:

The hardware specifications of the system used for this project are MacOS Big Sur, 8GB RAM, 1.8 GHz Dual-Core Intel Core i5. The steps used for installation are as follows:
Step 1: Firstly, download Hadoop and Spark packages with required version from the google.
Open terminal and check for the java version:
Java -version



```
Last login: Mon Jul  5 23:03:43 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) rohiths-MacBook-Air:~ rohithchityala$ java -version
java version "1.8.0_291"
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)
(base) rohiths-MacBook-Air:~ rohithchityala$
```

Fig 1.2. Java Version

Step 2:- Unzip the Spark package downloaded either manually or by using the command below:

tar -xzf spark-2.3.0-bin-hadoop2.7.tgz

Step 3:- Set up Java Home and Path. The below commands does the same.

export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_291.jdk/Contents/Home
PATH=$JAVA_HOME/bin:$PATH

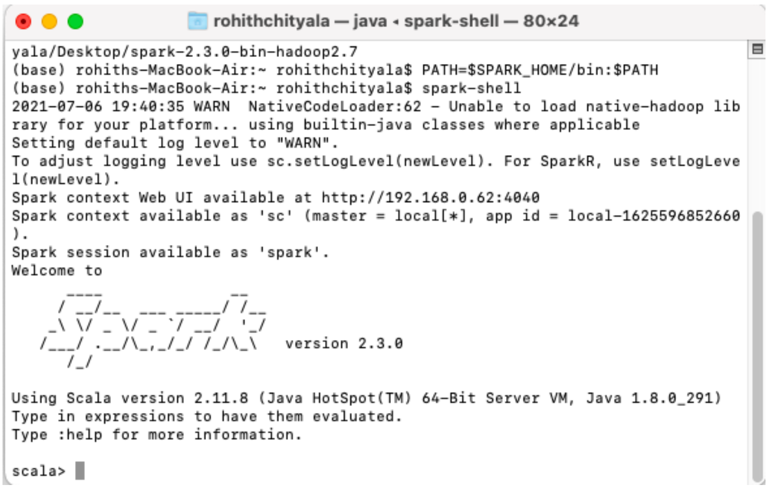
Step 4:- Setting up the Hadoop environment. The command for this is shown below:

export HADOOP_HOME=/Users/rohithchityala/Desktop/hadoop-2.7.3
PATH=$HADOOP_HOME/bin:$PATH

Step 5:- Set up the Spark path using below commands:

export SPARK_HOME=/Users/rohithchityala/Desktop/spark-2.3.0-bin-hadoop2.7
PATH=$SPARK_HOME/bin:$PATH

Step 5:- Type the command "spark-shell" and you will be able to see the output as below:

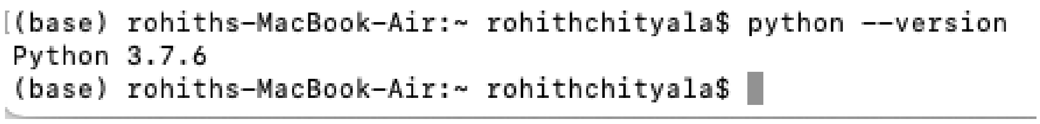


```
yala/Desktop/spark-2.3.0-bin-hadoop2.7
(base) rohiths-MacBook-Air:~ rohithchityala$ PATH=$SPARK_HOME/bin:$PATH
(base) rohiths-MacBook-Air:~ rohithchityala$ spark-shell
2021-07-06 19:40:35 WARN  NativeCodeLoader:62 - Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Spark context Web UI available at http://192.168.0.62:4040
Spark context available as 'sc' (master = local[*], app id = local-1625596852660
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_291)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Fig 1.3. Spark Version

Step 6:- Now, as we have installed Spark, it is time to launch PySpark. Before doing this it is important to check the python version available with us. Use the command below to check the version:

Python –version

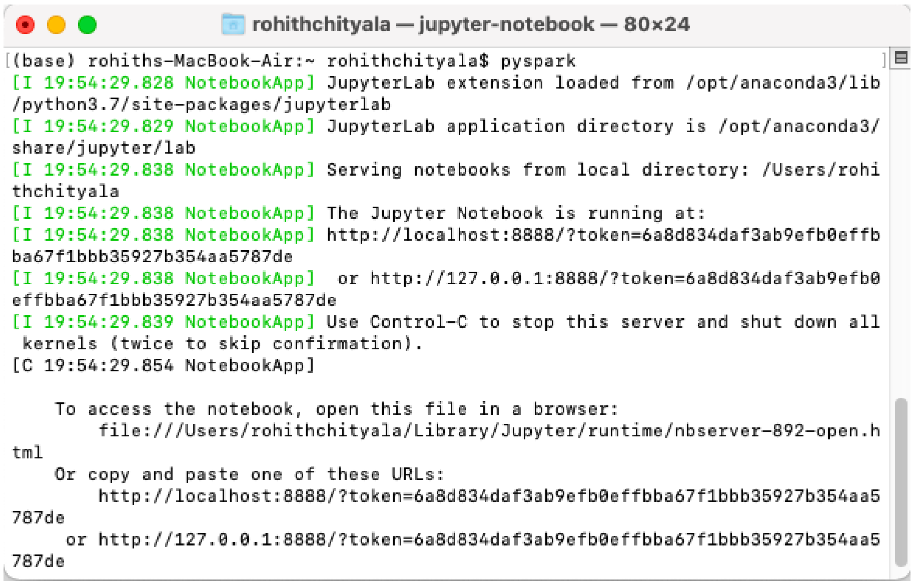


```
(base) rohiths-MacBook-Air:~ rohithchityala$ python --version
Python 3.7.6
(base) rohiths-MacBook-Air:~ rohithchityala$
```

Fig 1.4. Python Version

Step 7:- Finally, our main aim is to launch PySpark in Jupyter notebook. As I have Jupyter notebook in my system, I'm launching PySpark directly.

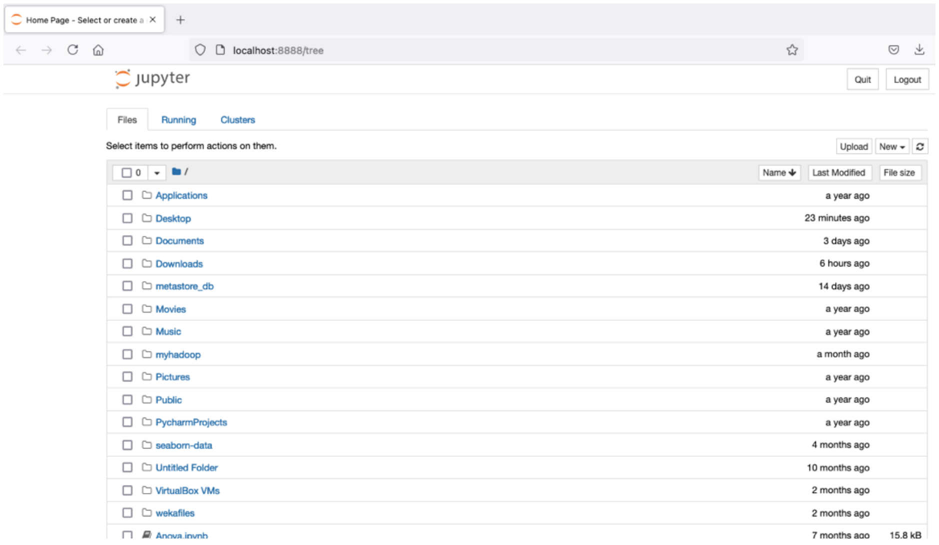
Pyspark



```
rohithchityala — jupyter-notebook — 80×24
(base) rohiths-MacBook-Air:~ rohithchityala$ pyspark
[I 19:54:29.828 NotebookApp] JupyterLab extension loaded from /opt/anaconda3/lib
/python3.7/site-packages/jupyterlab
[I 19:54:29.829 NotebookApp] JupyterLab application directory is /opt/anaconda3/
share/jupyter/lab
[I 19:54:29.838 NotebookApp] Serving notebooks from local directory: /Users/rohi
thchityala
[I 19:54:29.838 NotebookApp] The Jupyter Notebook is running at:
[I 19:54:29.838 NotebookApp] http://localhost:8888/?token=6a8d834daf3ab9efb0effb
ba67f1bbb35927b354aa5787de
[I 19:54:29.838 NotebookApp]  or http://127.0.0.1:8888/?token=6a8d834daf3ab9efb0
effbba67f1bbb35927b354aa5787de
[I 19:54:29.839 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 19:54:29.854 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///Users/rohithchityala/Library/Jupyter/runtime/nbserver-892-open.h
tml
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=6a8d834daf3ab9efb0effbba67f1bbb35927b354aa5
787de
     or http://127.0.0.1:8888/?token=6a8d834daf3ab9efb0effbba67f1bbb35927b354aa5
787de
```

Fig 1.5. Launching PySpark

Step 8:- Now, the jupyter notebook gets launched and we’re ready to use pyspark in jupyter. Also,



Select items to perform actions on them.

0 /	Name	Last Modified	File size
Applications		a year ago	
Desktop		23 minutes ago	
Documents		3 days ago	
Downloads		6 hours ago	
metastore_db		14 days ago	
Movies		a year ago	
Music		a year ago	
myhadoop		a month ago	
Pictures		a year ago	
Public		a year ago	
PycharmProjects		a year ago	
seaborn-data		4 months ago	
Untitled Folder		10 months ago	
VirtualBox VMs		2 months ago	
wekafiles		2 months ago	
Anova.ipynb		7 months ago	15.8 kB

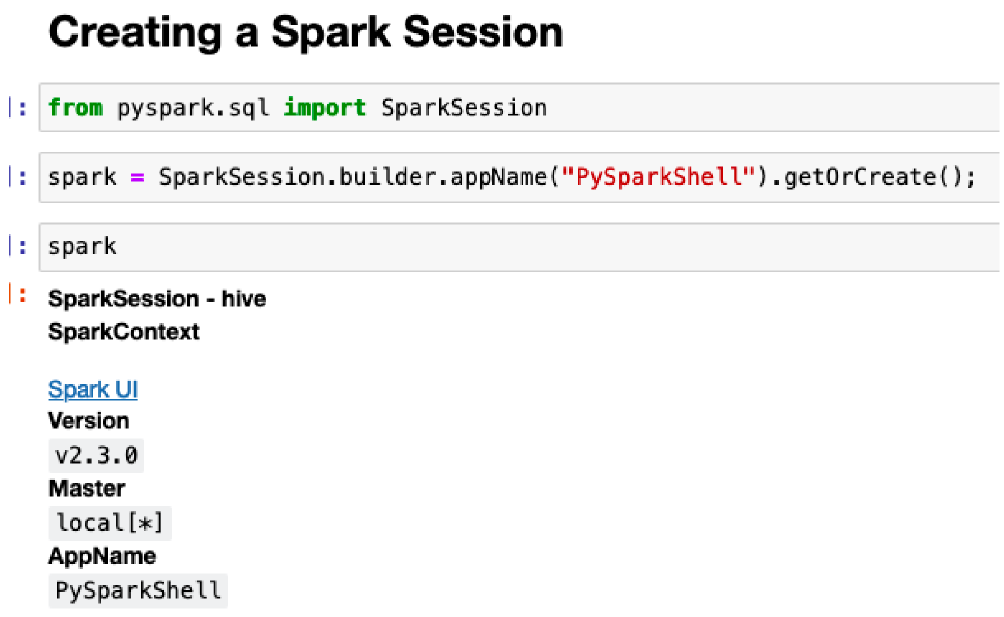
Fig 1.6. Jupyter Notebook

2.5 Processing of dataset in Jupyter notebook

In data analysis, data pre-processing is the essential and initial step to be performed. Pre-processing includes understanding, cleaning, and filtering the dataset.

2.5.1 Creating a SparkSession

Firstly, a SparkSession must be created, this can be done by the below mentioned code:



Creating a Spark Session

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName("PySparkShell").getOrCreate();
```

```
spark
```

SparkSession - hive

SparkContext

Spark UI

Version

`v2.3.0`

Master

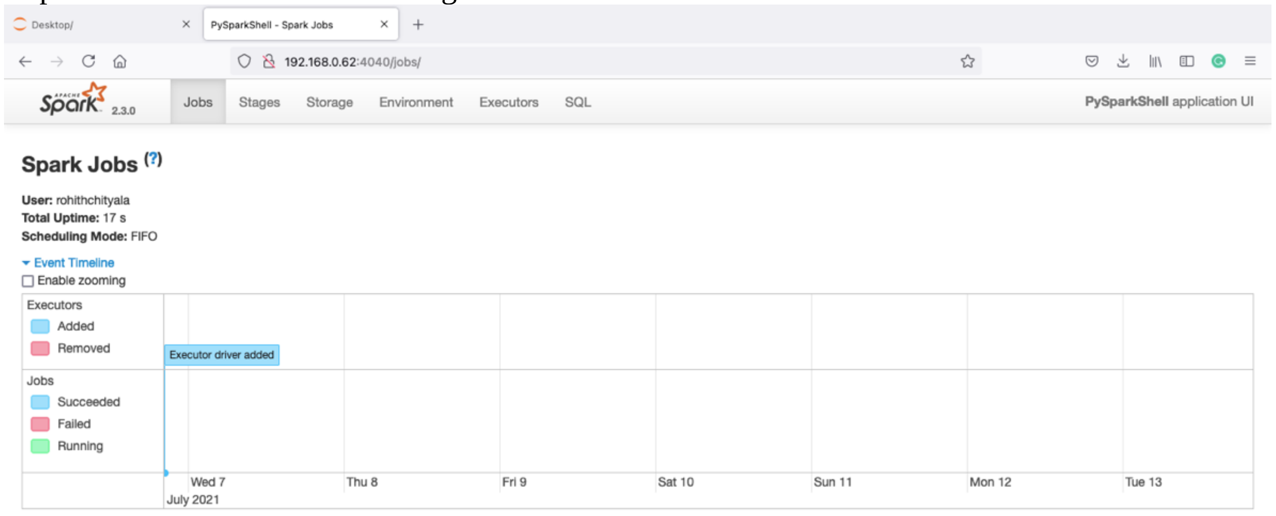
`local[*]`

AppName

`PySparkShell`

Fig 1.7. SparkSession

The SparkUI looks as shown in the figure below.



Spark Jobs (?)

User: rohithchityala
Total Uptime: 17 s
Scheduling Mode: FIFO

Event Timeline
Enable zooming

Fig 1.8. SparkUI

2.5.2 Loading the Dataset

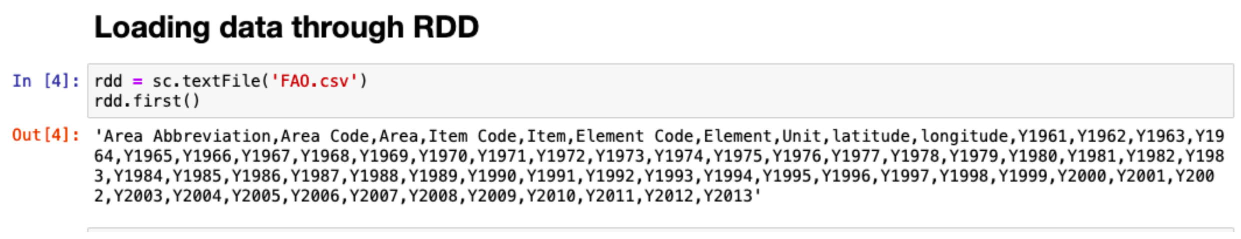
As we discussed in the above section, dataset can be loaded in two ways

Method 1: Loading the dataset using RDD:

The dataset will be loaded through RDD. The following code is used for this:

```
rdd = sc.textFile('FAO.csv')
rdd.first()
```

Here sc.textFile denotes SparkContext's textFile method which is used to load/create RDDs. First() is the operation applied on RDD which then returns the first element of the dataset as displayed below.



Loading data through RDD

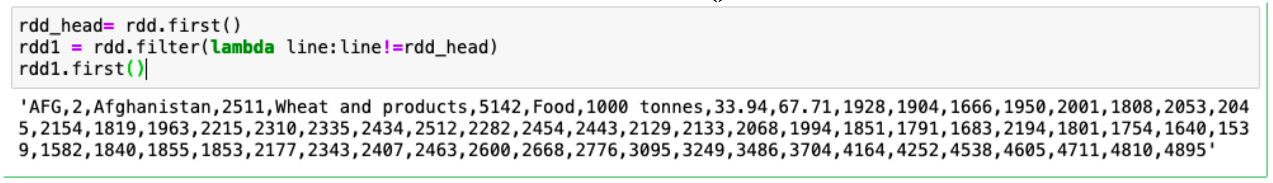
```
In [4]: rdd = sc.textFile('FAO.csv')
        rdd.first()

Out[4]: 'Area Abbreviation,Area Code,Area,Item Code,Item,Element Code,Element,Unit,latitude,longitude,Y1961,Y1962,Y1963,Y19
        64,Y1965,Y1966,Y1967,Y1968,Y1969,Y1970,Y1971,Y1972,Y1973,Y1974,Y1975,Y1976,Y1977,Y1978,Y1979,Y1980,Y1981,Y1982,Y198
        3,Y1984,Y1985,Y1986,Y1987,Y1988,Y1989,Y1990,Y1991,Y1992,Y1993,Y1994,Y1995,Y1996,Y1997,Y1998,Y1999,Y2000,Y2001,Y200
        2,Y2003,Y2004,Y2005,Y2006,Y2007,Y2008,Y2009,Y2010,Y2011,Y2012,Y2013'
```

Fig 1.9. RDD

2.5.3 Other Operations implemented on RDD

The below code returns the first row of values of all the columns present in the dataset.

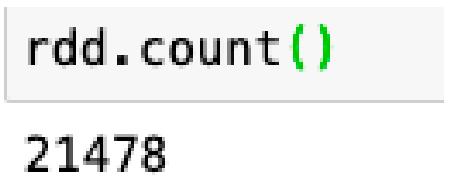
rdd_head= rdd.first()
rdd1 = rdd.filter(lambda line:line!=rdd_head)
rdd1.first()



```
rdd_head= rdd.first()
rdd1 = rdd.filter(lambda line:line!=rdd_head)
rdd1.first()

'AFG,2,Afghanistan,2511,Wheat and products,5142,Food,1000 tonnes,33.94,67.71,1928,1904,1666,1950,2001,1808,2053,204
5,2154,1819,1963,2215,2310,2335,2434,2512,2282,2454,2443,2129,2133,2068,1994,1851,1791,1683,2194,1801,1754,1640,153
9,1582,1840,1855,1853,2177,2343,2407,2463,2600,2668,2776,3095,3249,3486,3704,4164,4252,4538,4605,4711,4810,4895'
```

Fig 1.10. Operations on RDD

By applying Count() function on the RDD, gives us the total number of elements present in the dataset. The results are as show below:



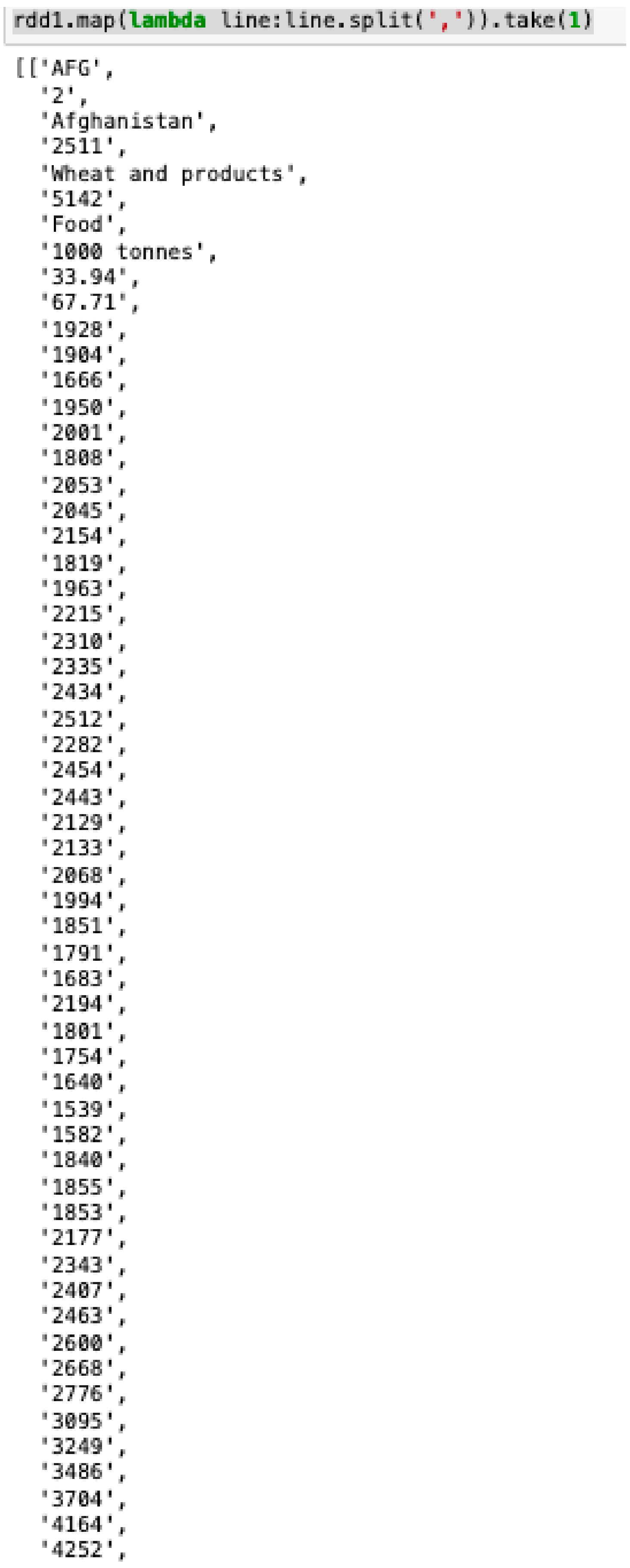
```
rdd.count()

21478
```

Fig 1.11. Count() function

Map function() is the transformation which when applied on an RDD, returns the data where all the elements are separated by comma. The code for this transformation is:

rdd1.map(lambda line:line.split(',')).take(1)

The results can be seen in below figure.



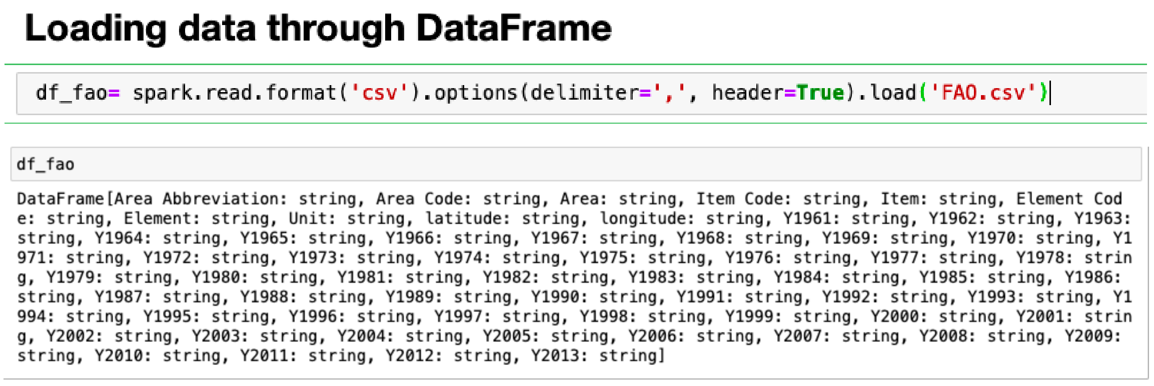
```
rdd1.map(lambda line:line.split(',')).take(1)
```

```
[['AFG',
  '2',
  'Afghanistan',
  '2511',
  'Wheat and products',
  '5142',
  'Food',
  '1000 tonnes',
  '33.94',
  '67.71',
  '1928',
  '1904',
  '1666',
  '1950',
  '2001',
  '1808',
  '2053',
  '2045',
  '2154',
  '1819',
  '1963',
  '2215',
  '2310',
  '2335',
  '2434',
  '2512',
  '2282',
  '2454',
  '2443',
  '2129',
  '2133',
  '2068',
  '1994',
  '1851',
  '1791',
  '1683',
  '2194',
  '1801',
  '1754',
  '1640',
  '1539',
  '1582',
  '1840',
  '1855',
  '1853',
  '2177',
  '2343',
  '2407',
  '2463',
  '2600',
  '2668',
  '2776',
  '3095',
  '3249',
  '3486',
  '3704',
  '4164',
  '4252',
```

Fig 1.12. Map Function

2.5.4 Loading data through DataFrame:

The below code shows how data can be loaded through a dataframe.



Loading data through DataFrame

```
df_fao= spark.read.format('csv').options(delimiter=',', header=True).load('FAO.csv')
```

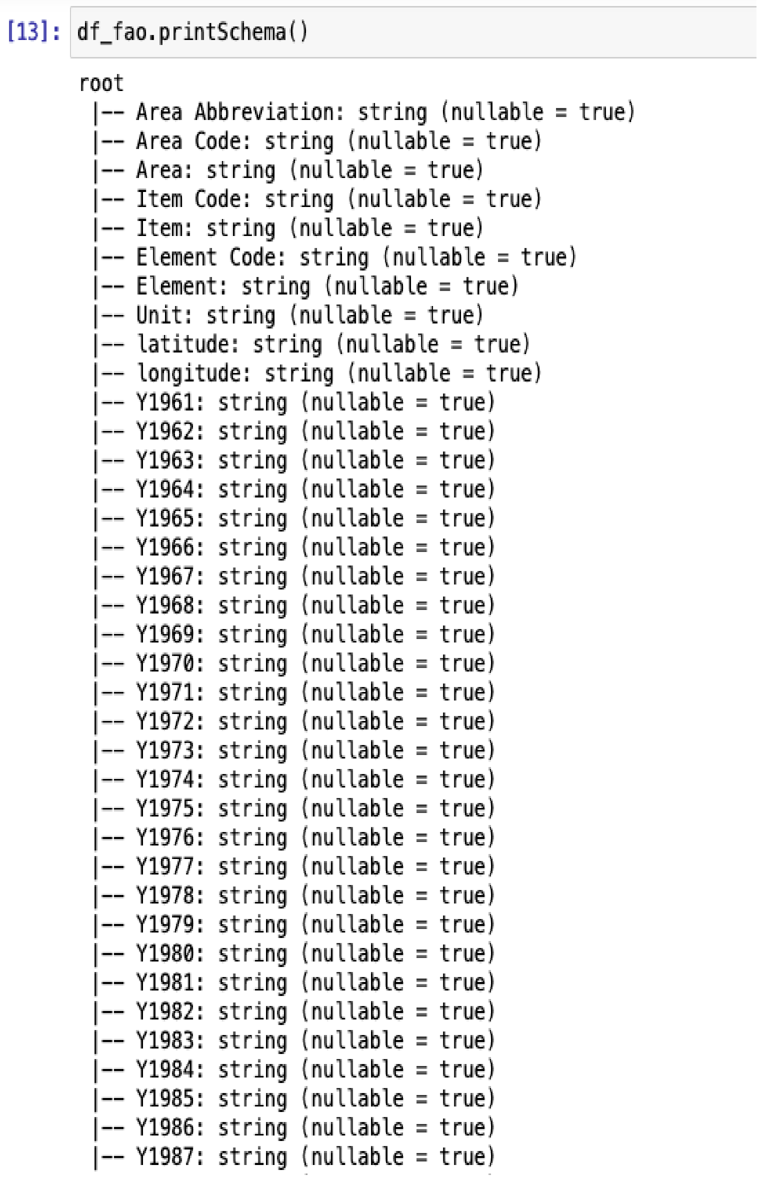
```
df_fao
```

```
DataFrame[Area Abbreviation: string, Area Code: string, Area: string, Item Code: string, Item: string, Element Cod
e: string, Element: string, Unit: string, latitude: string, longitude: string, Y1961: string, Y1962: string, Y1963:
string, Y1964: string, Y1965: string, Y1966: string, Y1967: string, Y1968: string, Y1969: string, Y1970: string, Y1
971: string, Y1972: string, Y1973: string, Y1974: string, Y1975: string, Y1976: string, Y1977: string, Y1978: strin
g, Y1979: string, Y1980: string, Y1981: string, Y1982: string, Y1983: string, Y1984: string, Y1985: string, Y1986:
string, Y1987: string, Y1988: string, Y1989: string, Y1990: string, Y1991: string, Y1992: string, Y1993: string, Y1
994: string, Y1995: string, Y1996: string, Y1997: string, Y1998: string, Y1999: string, Y2000: string, Y2001: strin
g, Y2002: string, Y2003: string, Y2004: string, Y2005: string, Y2006: string, Y2007: string, Y2008: string, Y2009:
string, Y2010: string, Y2011: string, Y2012: string, Y2013: string]
```

Fig 1.13. Loading the dataset

2.5.5. Finding the datatypes.

The below code is used for this:

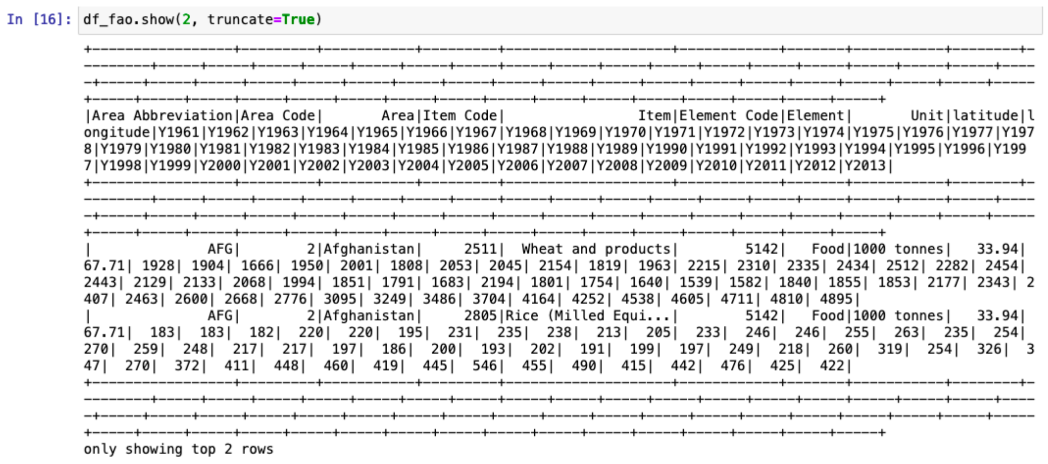


```
[13]: df_fao.printSchema()

root
 |-- Area Abbreviation: string (nullable = true)
 |-- Area Code: string (nullable = true)
 |-- Area: string (nullable = true)
 |-- Item Code: string (nullable = true)
 |-- Item: string (nullable = true)
 |-- Element Code: string (nullable = true)
 |-- Element: string (nullable = true)
 |-- Unit: string (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
 |-- Y1961: string (nullable = true)
 |-- Y1962: string (nullable = true)
 |-- Y1963: string (nullable = true)
 |-- Y1964: string (nullable = true)
 |-- Y1965: string (nullable = true)
 |-- Y1966: string (nullable = true)
 |-- Y1967: string (nullable = true)
 |-- Y1968: string (nullable = true)
 |-- Y1969: string (nullable = true)
 |-- Y1970: string (nullable = true)
 |-- Y1971: string (nullable = true)
 |-- Y1972: string (nullable = true)
 |-- Y1973: string (nullable = true)
 |-- Y1974: string (nullable = true)
 |-- Y1975: string (nullable = true)
 |-- Y1976: string (nullable = true)
 |-- Y1977: string (nullable = true)
 |-- Y1978: string (nullable = true)
 |-- Y1979: string (nullable = true)
 |-- Y1980: string (nullable = true)
 |-- Y1981: string (nullable = true)
 |-- Y1982: string (nullable = true)
 |-- Y1983: string (nullable = true)
 |-- Y1984: string (nullable = true)
 |-- Y1985: string (nullable = true)
 |-- Y1986: string (nullable = true)
 |-- Y1987: string (nullable = true)
```

Fig 1.14. printSchema()

2.5.6 Show() function:

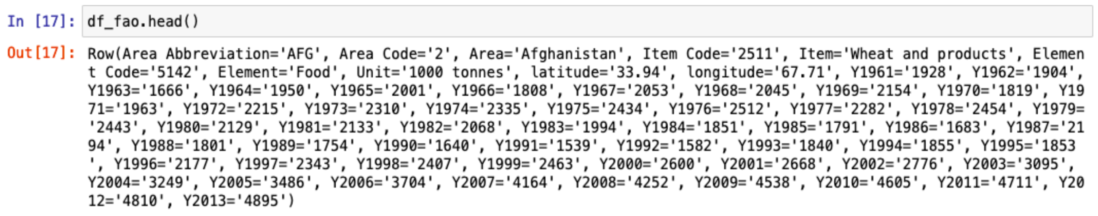
The below shown code is used to display the first two elements of the dataset:



```
In [16]: df_fao.show(2, truncate=True)

+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Area Abbreviation|Area Code|       Area|Item Code|                Item|Element Code|Element|       Unit|latitude|l
ongitude|Y1961|Y1962|Y1963|Y1964|Y1965|Y1966|Y1967|Y1968|Y1969|Y1970|Y1971|Y1972|Y1973|Y1974|Y1975|Y1976|Y1977|Y197
8|Y1979|Y1980|Y1981|Y1982|Y1983|Y1984|Y1985|Y1986|Y1987|Y1988|Y1989|Y1990|Y1991|Y1992|Y1993|Y1994|Y1995|Y1996|Y199
7|Y1998|Y1999|Y2000|Y2001|Y2002|Y2003|Y2004|Y2005|Y2006|Y2007|Y2008|Y2009|Y2010|Y2011|Y2012|Y2013|
+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|              AFG|        2|Afghanistan|     2511|  Wheat and products|        5142|   Food|1000 tonnes|   33.94|
67.71| 1928| 1904| 1666| 1950| 2001| 1808| 2053| 2045| 2154| 1819| 1963| 2215| 2310| 2335| 2434| 2512| 2282| 2454|
2443| 2129| 2133| 2068| 1994| 1851| 1791| 1683| 2194| 1801| 1754| 1640| 1539| 1582| 1840| 1855| 1853| 2177| 2343| 2
407| 2463| 2600| 2668| 2776| 3095| 3249| 3486| 3704| 4164| 4252| 4538| 4605| 4711| 4810| 4895|
|              AFG|        2|Afghanistan|     2805|Rice (Milled Equi...|        5142|   Food|1000 tonnes|   33.94|
67.71|  183|  183|  182|  220|  220|  195|  231|  235|  238|  213|  205|  233|  246|  246|  255|  263|  235|  254|
270|  259|  248|  217|  217|  197|  186|  200|  193|  202|  191|  199|  197|  249|  218|  260|  319|  254|  326|  3
47|  270|  372|  411|  448|  460|  419|  445|  546|  455|  490|  415|  442|  476|  425|  422|
+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 2 rows
```

Fig 1.15. Show() function

2.5.7 Head() Function



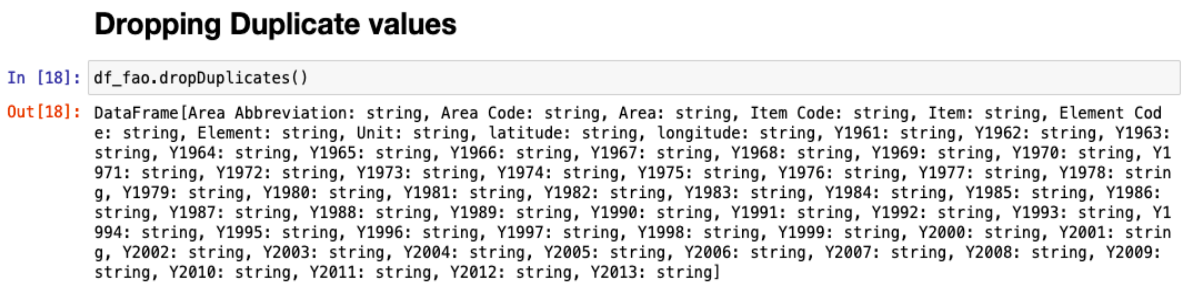
```
In [17]: df_fao.head()

Out[17]: Row(Area Abbreviation='AFG', Area Code='2', Area='Afghanistan', Item Code='2511', Item='Wheat and products', Elemen
t Code='5142', Element='Food', Unit='1000 tonnes', latitude='33.94', longitude='67.71', Y1961='1928', Y1962='1904',
Y1963='1666', Y1964='1950', Y1965='2001', Y1966='1808', Y1967='2053', Y1968='2045', Y1969='2154', Y1970='1819', Y19
71='1963', Y1972='2215', Y1973='2310', Y1974='2335', Y1975='2434', Y1976='2512', Y1977='2282', Y1978='2454', Y1979=
'2443', Y1980='2129', Y1981='2133', Y1982='2068', Y1983='1994', Y1984='1851', Y1985='1791', Y1986='1683', Y1987='21
94', Y1988='1801', Y1989='1754', Y1990='1640', Y1991='1539', Y1992='1582', Y1993='1840', Y1994='1855', Y1995='1853
', Y1996='2177', Y1997='2343', Y1998='2407', Y1999='2463', Y2000='2600', Y2001='2668', Y2002='2776', Y2003='3095',
Y2004='3249', Y2005='3486', Y2006='3704', Y2007='4164', Y2008='4252', Y2009='4538', Y2010='4605', Y2011='4711', Y20
12='4810', Y2013='4895')
```

Fig 1.16. Head() function

2.5.8 Dropping duplicate values

In a dataset it is important to check and drop duplicate values which

makes the data clean. The below code does this.



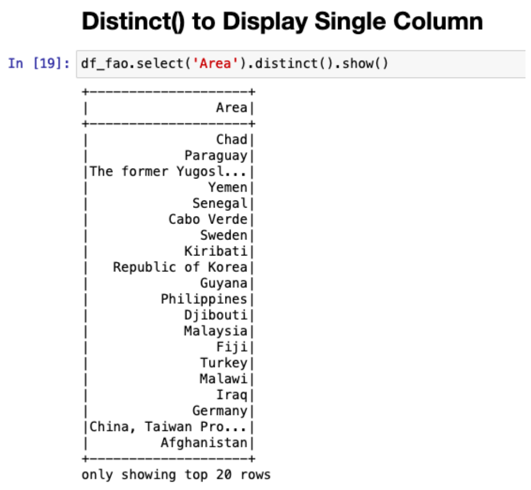
Dropping Duplicate values

```
In [18]: df_fao.dropDuplicates()

Out[18]: DataFrame[Area Abbreviation: string, Area Code: string, Area: string, Item Code: string, Item: string, Element Cod
e: string, Element: string, Unit: string, latitude: string, longitude: string, Y1961: string, Y1962: string, Y1963:
string, Y1964: string, Y1965: string, Y1966: string, Y1967: string, Y1968: string, Y1969: string, Y1970: string, Y1
971: string, Y1972: string, Y1973: string, Y1974: string, Y1975: string, Y1976: string, Y1977: string, Y1978: strin
g, Y1979: string, Y1980: string, Y1981: string, Y1982: string, Y1983: string, Y1984: string, Y1985: string, Y1986:
string, Y1987: string, Y1988: string, Y1989: string, Y1990: string, Y1991: string, Y1992: string, Y1993: string, Y1
994: string, Y1995: string, Y1996: string, Y1997: string, Y1998: string, Y1999: string, Y2000: string, Y2001: strin
g, Y2002: string, Y2003: string, Y2004: string, Y2005: string, Y2006: string, Y2007: string, Y2008: string, Y2009:
string, Y2010: string, Y2011: string, Y2012: string, Y2013: string]
```

Fig 1.17. Dropping values

2.5.8 Distinct Function –

It is used to display selected/single column. The below code shows this.
Here I used distinct() function to display Area column as an example.



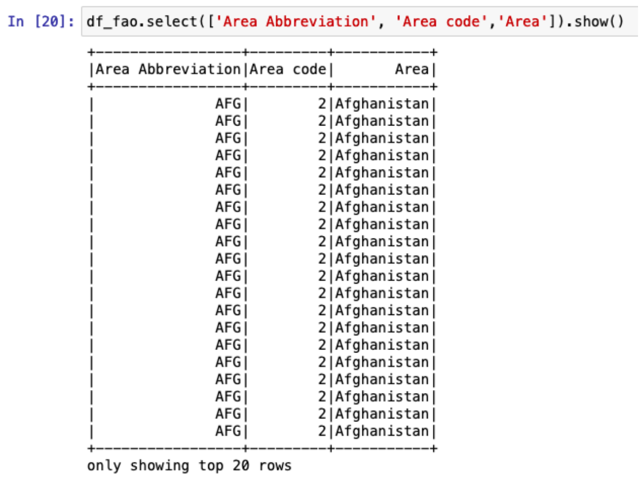
Distinct() to Display Single Column

```
In [19]: df_fao.select('Area').distinct().show()

+--------------------+
|                Area|
+--------------------+
|                Chad|
|            Paraguay|
|The former Yugosl...|
|               Yemen|
|             Senegal|
|          Cabo Verde|
|              Sweden|
|            Kiribati|
|   Republic of Korea|
|              Guyana|
|         Philippines|
|            Djibouti|
|            Malaysia|
|                Fiji|
|              Turkey|
|              Malawi|
|                Iraq|
|             Germany|
|China, Taiwan Pro...|
|         Afghanistan|
+--------------------+
only showing top 20 rows
```

Fig 1.18. Distinct()

2.5.9 Select Function

This function is used to display selected columns of a dataset.

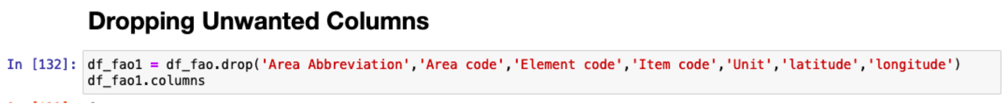


```
In [20]: df_fao.select(['Area Abbreviation', 'Area code','Area']).show()

+-----------------+---------+-----------+
|Area Abbreviation|Area code|       Area|
+-----------------+---------+-----------+
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
|              AFG|        2|Afghanistan|
+-----------------+---------+-----------+
only showing top 20 rows
```

Fig 1.19. Select()

2.5.10 Dropping unwanted columns

The below code shows how to drop unwanted columns.

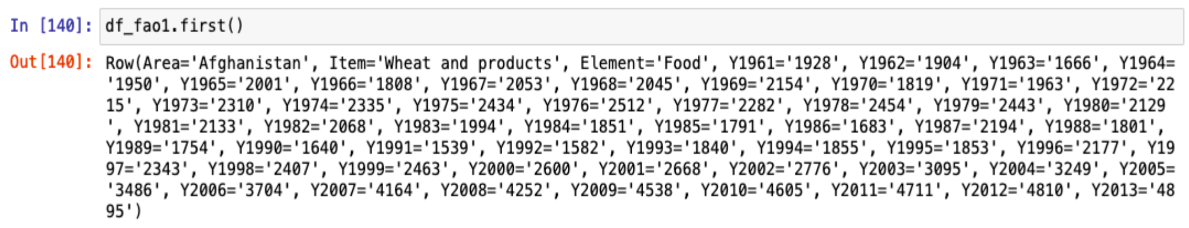


Dropping Unwanted Columns

```
In [132]: df_fao1 = df_fao.drop('Area Abbreviation','Area code','Element code','Item code','Unit','latitude','longitude')
          df_fao1.columns
```

Fig 1.20. Unwanted columns

The data obtained after dropping unwanted columns is assigned to a new variable df_fao1 and it is displayed as shown below.

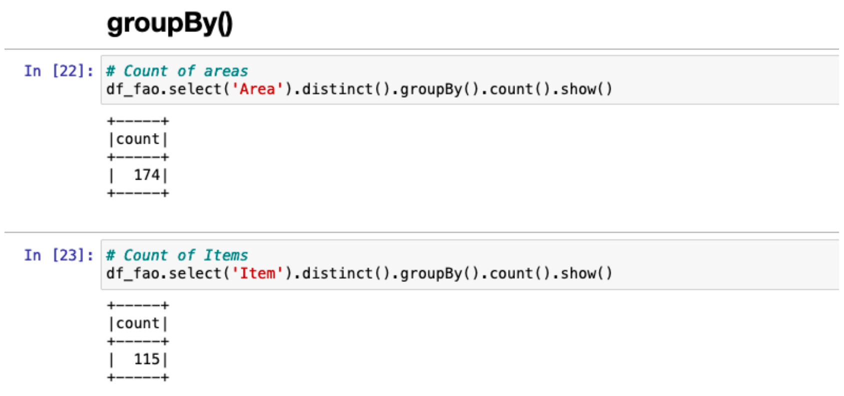


```
In [140]: df_fao1.first()

Out[140]: Row(Area='Afghanistan', Item='Wheat and products', Element='Food', Y1961='1928', Y1962='1904', Y1963='1666', Y1964=
          '1950', Y1965='2001', Y1966='1808', Y1967='2053', Y1968='2045', Y1969='2154', Y1970='1819', Y1971='1963', Y1972='22
          15', Y1973='2310', Y1974='2335', Y1975='2434', Y1976='2512', Y1977='2282', Y1978='2454', Y1979='2443', Y1980='2129
          ', Y1981='2133', Y1982='2068', Y1983='1994', Y1984='1851', Y1985='1791', Y1986='1683', Y1987='2194', Y1988='1801',
          Y1989='1754', Y1990='1640', Y1991='1539', Y1992='1582', Y1993='1840', Y1994='1855', Y1995='1853', Y1996='2177', Y19
          97='2343', Y1998='2407', Y1999='2463', Y2000='2600', Y2001='2668', Y2002='2776', Y2003='3095', Y2004='3249', Y2005=
          '3486', Y2006='3704', Y2007='4164', Y2008='4252', Y2009='4538', Y2010='4605', Y2011='4711', Y2012='4810', Y2013='48
          95')
```

Fig 1.21. First()

2.5.11 GroupBy()

This function is used to perform operations on a specific column as shown below.



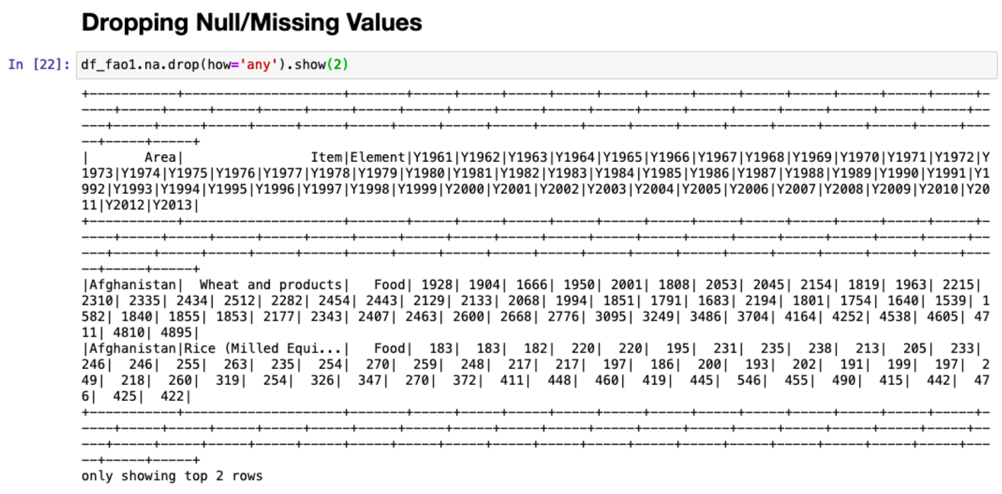
groupBy()

```
In [22]: # Count of areas
         df_fao.select('Area').distinct().groupBy().count().show()

         +-----+
         |count|
         +-----+
         |  174|
         +-----+

In [23]: # Count of Items
         df_fao.select('Item').distinct().groupBy().count().show()

         +-----+
         |count|
         +-----+
         |  115|
         +-----+
```

Fig 1.22. GroupBy()

2.5.12 Dropping missing values

In this part, missing values present in the dataset are dropped using the code below.

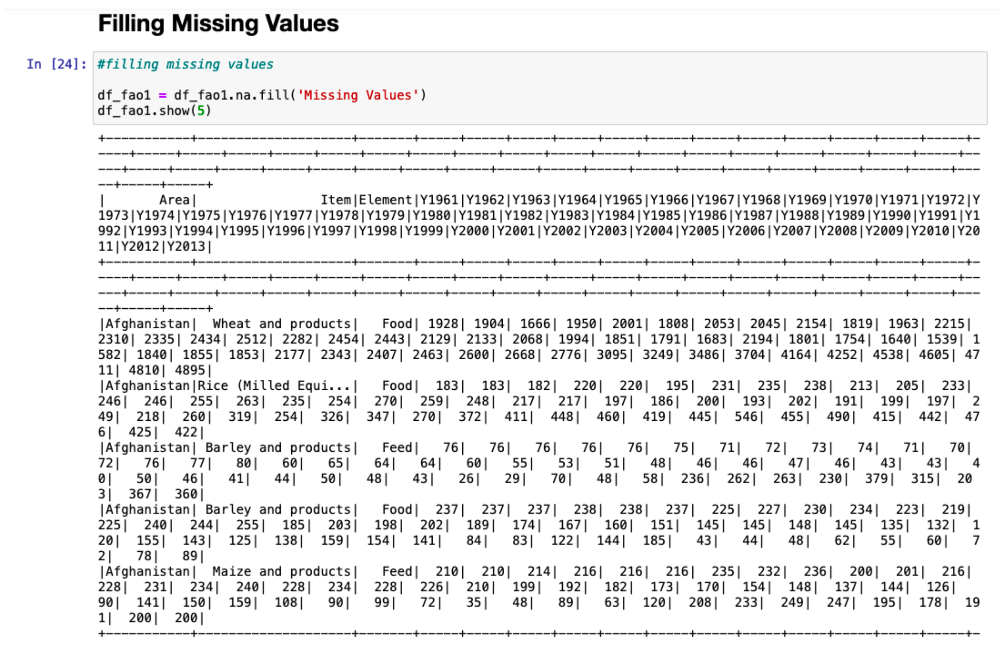


Dropping Null/Missing Values

```
In [22]: df_fao1.na.drop(how='any').show(2)

+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+
|       Area|                Item|Element|Y1961|Y1962|Y1963|Y1964|Y1965|Y1966|Y1967|Y1968|Y1969|Y1970|Y1971|Y1972|Y
1973|Y1974|Y1975|Y1976|Y1977|Y1978|Y1979|Y1980|Y1981|Y1982|Y1983|Y1984|Y1985|Y1986|Y1987|Y1988|Y1989|Y1990|Y1991|Y1
992|Y1993|Y1994|Y1995|Y1996|Y1997|Y1998|Y1999|Y2000|Y2001|Y2002|Y2003|Y2004|Y2005|Y2006|Y2007|Y2008|Y2009|Y2010|Y20
11|Y2012|Y2013|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+
|Afghanistan|  Wheat and products|   Food| 1928| 1904| 1666| 1950| 2001| 1808| 2053| 2045| 2154| 1819| 1963| 2215|
2310| 2335| 2434| 2512| 2282| 2454| 2443| 2129| 2133| 2068| 1994| 1851| 1791| 1683| 2194| 1801| 1754| 1640| 1539| 1
582| 1840| 1855| 1853| 2177| 2343| 2407| 2463| 2600| 2668| 2776| 3095| 3249| 3486| 3704| 4164| 4252| 4538| 4605| 47
11| 4810| 4895|
|Afghanistan|Rice (Milled Equi...|   Food|  183|  183|  182|  220|  220|  195|  231|  235|  238|  213|  205|  233|
246|  246|  255|  263|  235|  254|  270|  259|  248|  217|  217|  197|  186|  200|  193|  202|  191|  199|  197|  2
49|  218|  260|  319|  254|  326|  347|  270|  372|  411|  448|  460|  419|  445|  546|  455|  490|  415|  442|  47
6|  425|  422|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+
only showing top 2 rows
```

Fig 1.23. Dropping missing values

Now, as we dropped missing values, it is also important to fill the missing values. This is done using the below code:



Filling Missing Values

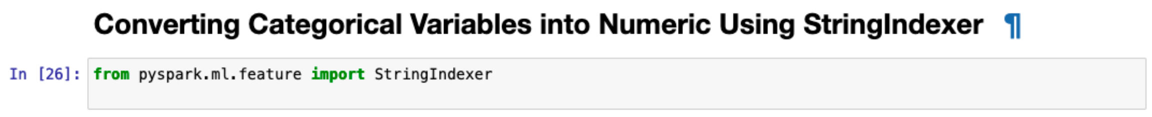
```
In [24]: #filling missing values

         df_fao1 = df_fao1.na.fill('Missing Values')
         df_fao1.show(5)
```

```
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+
|       Area|                Item|Element|Y1961|Y1962|Y1963|Y1964|Y1965|Y1966|Y1967|Y1968|Y1969|Y1970|Y1971|Y1972|Y
1973|Y1974|Y1975|Y1976|Y1977|Y1978|Y1979|Y1980|Y1981|Y1982|Y1983|Y1984|Y1985|Y1986|Y1987|Y1988|Y1989|Y1990|Y1991|Y1
992|Y1993|Y1994|Y1995|Y1996|Y1997|Y1998|Y1999|Y2000|Y2001|Y2002|Y2003|Y2004|Y2005|Y2006|Y2007|Y2008|Y2009|Y2010|Y20
11|Y2012|Y2013|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+
|Afghanistan|  Wheat and products|   Food| 1928| 1904| 1666| 1950| 2001| 1808| 2053| 2045| 2154| 1819| 1963| 2215|
2310| 2335| 2434| 2512| 2282| 2454| 2443| 2129| 2133| 2068| 1994| 1851| 1791| 1683| 2194| 1801| 1754| 1640| 1539| 1
582| 1840| 1855| 1853| 2177| 2343| 2407| 2463| 2600| 2668| 2776| 3095| 3249| 3486| 3704| 4164| 4252| 4538| 4605| 47
11| 4810| 4895|
|Afghanistan|Rice (Milled Equi...|   Food|  183|  183|  182|  220|  220|  195|  231|  235|  238|  213|  205|  233|
246|  246|  255|  263|  235|  254|  270|  259|  248|  217|  217|  197|  186|  200|  193|  202|  191|  199|  197|  2
49|  218|  260|  319|  254|  326|  347|  270|  372|  411|  448|  460|  419|  445|  546|  455|  490|  415|  442|  47
6|  425|  422|
|Afghanistan| Barley and products|   Feed|   76|   76|   76|   76|   76|   75|   71|   72|   73|   74|   71|   70|
72|   76|   77|   80|   60|   65|   64|   64|   60|   55|   53|   51|   48|   46|   46|   47|   46|   43|   43|   4
0|   50|   46|   41|   44|   50|   48|   43|   26|   29|   70|   48|   58|  236|  262|  263|  230|  379|  315|  20
3|  367|  360|
|Afghanistan| Barley and products|   Food|  237|  237|  237|  238|  238|  237|  225|  227|  230|  234|  223|  219|
225|  240|  244|  255|  185|  203|  198|  202|  189|  174|  167|  160|  151|  145|  145|  148|  145|  135|  132|  1
20|  155|  143|  125|  138|  159|  154|  141|   84|   83|  122|  144|  185|   43|   44|   48|   62|   55|   60|   7
2|   78|   89|
|Afghanistan|  Maize and products|   Feed|  210|  210|  214|  216|  216|  216|  235|  232|  236|  200|  201|  216|
228|  231|  234|  240|  228|  234|  228|  226|  210|  199|  192|  182|  173|  170|  154|  148|  137|  144|  126|
90|  141|  150|  159|  108|   90|   99|   72|   35|   48|   89|   63|  120|  208|  233|  249|  247|  195|  178|  19
1|  200|  200|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
```

Fig 1.24. Filling missing values

2.6 StringIndexer

This operation is used to convert categorical variables to numerical variables and the code is shown below.

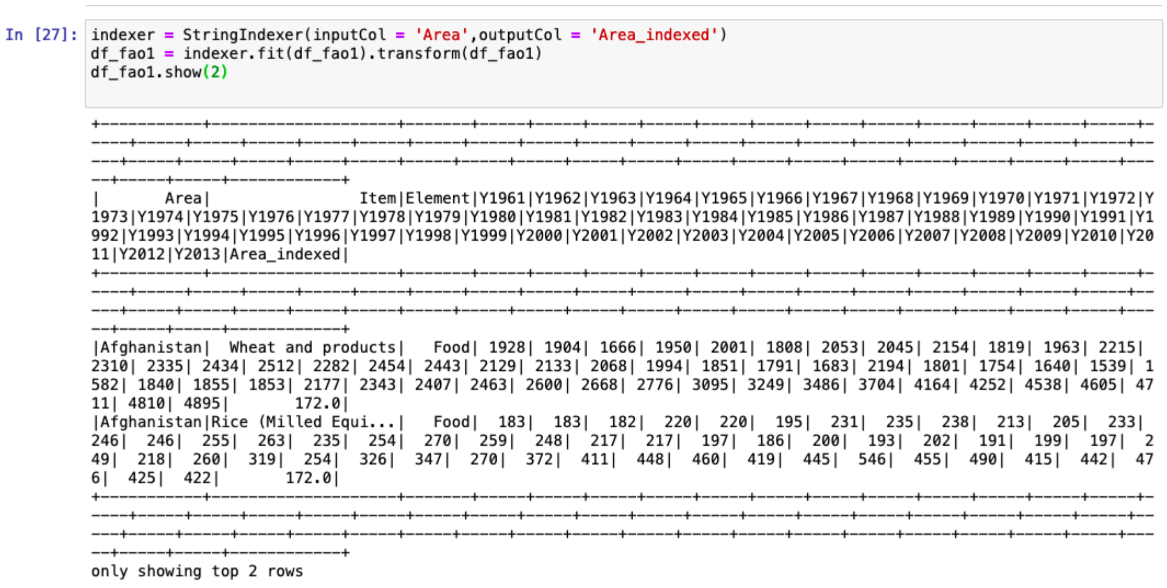


Converting Categorical Variables into Numeric Using StringIndexer ¶

```
In [26]: from pyspark.ml.feature import StringIndexer
```

Fig 1.25. StringIndexer

For example, as shown in the below code, I have converted column 'Area' into numerical variable 'Area_indexed'.

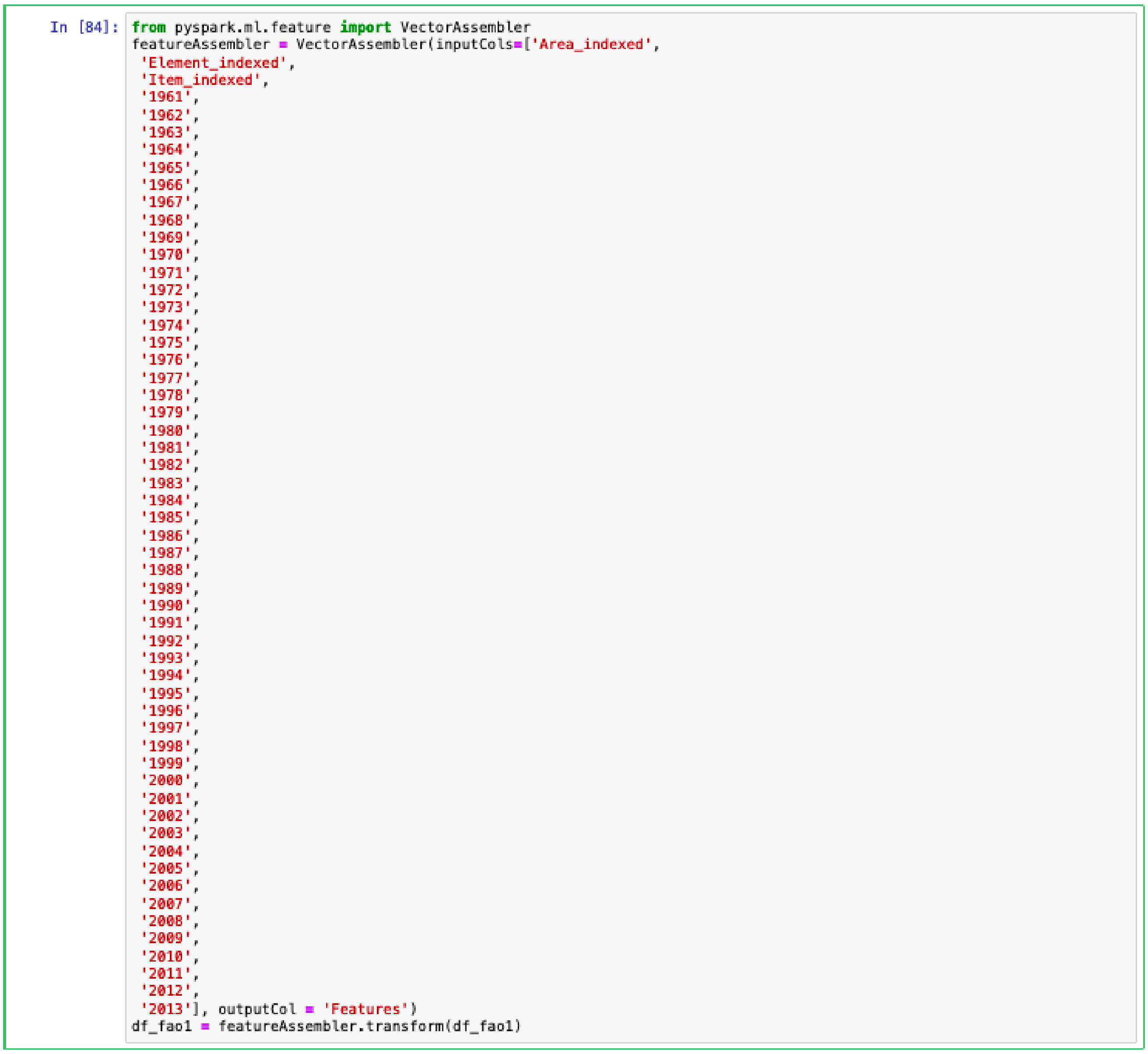


```
In [27]: indexer = StringIndexer(inputCol = 'Area',outputCol = 'Area_indexed')
         df_fao1 = indexer.fit(df_fao1).transform(df_fao1)
         df_fao1.show(2)
```

```
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+------------+
|       Area|                Item|Element|Y1961|Y1962|Y1963|Y1964|Y1965|Y1966|Y1967|Y1968|Y1969|Y1970|Y1971|Y1972|Y
1973|Y1974|Y1975|Y1976|Y1977|Y1978|Y1979|Y1980|Y1981|Y1982|Y1983|Y1984|Y1985|Y1986|Y1987|Y1988|Y1989|Y1990|Y1991|Y1
992|Y1993|Y1994|Y1995|Y1996|Y1997|Y1998|Y1999|Y2000|Y2001|Y2002|Y2003|Y2004|Y2005|Y2006|Y2007|Y2008|Y2009|Y2010|Y20
11|Y2012|Y2013|Area_indexed|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+------------+
|Afghanistan|  Wheat and products|   Food| 1928| 1904| 1666| 1950| 2001| 1808| 2053| 2045| 2154| 1819| 1963| 2215|
2310| 2335| 2434| 2512| 2282| 2454| 2443| 2129| 2133| 2068| 1994| 1851| 1791| 1683| 2194| 1801| 1754| 1640| 1539| 1
582| 1840| 1855| 1853| 2177| 2343| 2407| 2463| 2600| 2668| 2776| 3095| 3249| 3486| 3704| 4164| 4252| 4538| 4605| 47
11| 4810| 4895|       172.0|
|Afghanistan|Rice (Milled Equi...|   Food|  183|  183|  182|  220|  220|  195|  231|  235|  238|  213|  205|  233|
246|  246|  255|  263|  235|  254|  270|  259|  248|  217|  217|  197|  186|  200|  193|  202|  191|  199|  197|  2
49|  218|  260|  319|  254|  326|  347|  270|  372|  411|  448|  460|  419|  445|  546|  455|  490|  415|  442|  47
6|  425|  422|       172.0|
+-----------+--------------------+-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+---
--+-----+-----+------------+
only showing top 2 rows
```

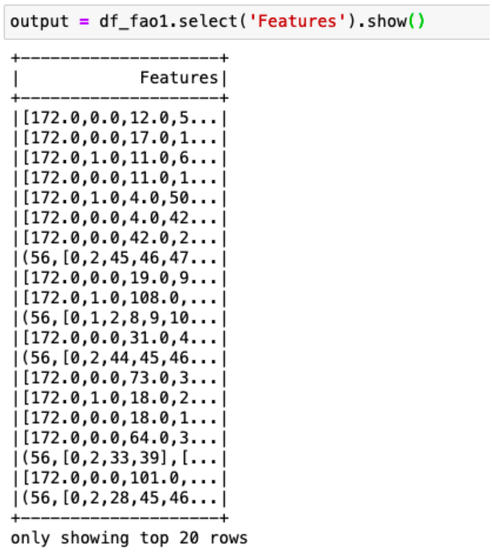
Fig 1.26. StringIndexer for Area column

Now, as we created all the numeric variables, I created a column 'Features' which consists of all indexed variables as shown below.



```
In [84]: from pyspark.ml.feature import VectorAssembler
         featureAssembler = VectorAssembler(inputCols=['Area_indexed',
          'Element_indexed',
          'Item_indexed',
          '1961',
          '1962',
          '1963',
          '1964',
          '1965',
          '1966',
          '1967',
          '1968',
          '1969',
          '1970',
          '1971',
          '1972',
          '1973',
          '1974',
          '1975',
          '1976',
          '1977',
          '1978',
          '1979',
          '1980',
          '1981',
          '1982',
          '1983',
          '1984',
          '1985',
          '1986',
          '1987',
          '1988',
          '1989',
          '1990',
          '1991',
          '1992',
          '1993',
          '1994',
          '1995',
          '1996',
          '1997',
          '1998',
          '1999',
          '2000',
          '2001',
          '2002',
          '2003',
          '2004',
          '2005',
          '2006',
          '2007',
          '2008',
          '2009',
          '2010',
          '2011',
          '2012',
          '2013'], outputCol = 'Features')
         df_fao1 = featureAssembler.transform(df_fao1)
```

Fig 1.27. VectorAssembler



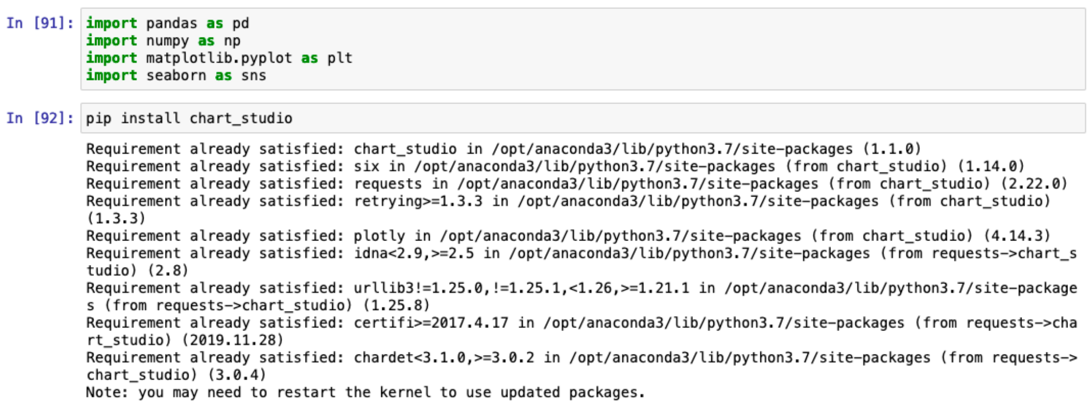
```
output = df_fao1.select('Features').show()

+--------------------+
|            Features|
+--------------------+
|[172.0,0.0,12.0,5...|
|[172.0,0.0,17.0,1...|
|[172.0,1.0,11.0,6...|
|[172.0,0.0,11.0,1...|
|[172.0,1.0,4.0,50...|
|[172.0,0.0,4.0,42...|
|[172.0,0.0,42.0,2...|
|(56,[0,2,45,46,47...|
|[172.0,0.0,19.0,9...|
|[172.0,1.0,108.0,...|
|(56,[0,1,2,8,9,10...|
|[172.0,0.0,31.0,4...|
|(56,[0,2,44,45,46...|
|[172.0,0.0,73.0,3...|
|[172.0,1.0,18.0,2...|
|[172.0,0.0,18.0,1...|
|[172.0,0.0,64.0,3...|
|(56,[0,2,33,39],[...|
|[172.0,0.0,101.0,...|
|(56,[0,2,28,45,46...|
+--------------------+
only showing top 20 rows
```

Fig 1.28. Features column

In the above code, the newly created features have been displayed.

2.7 Exploratory Data Analysis

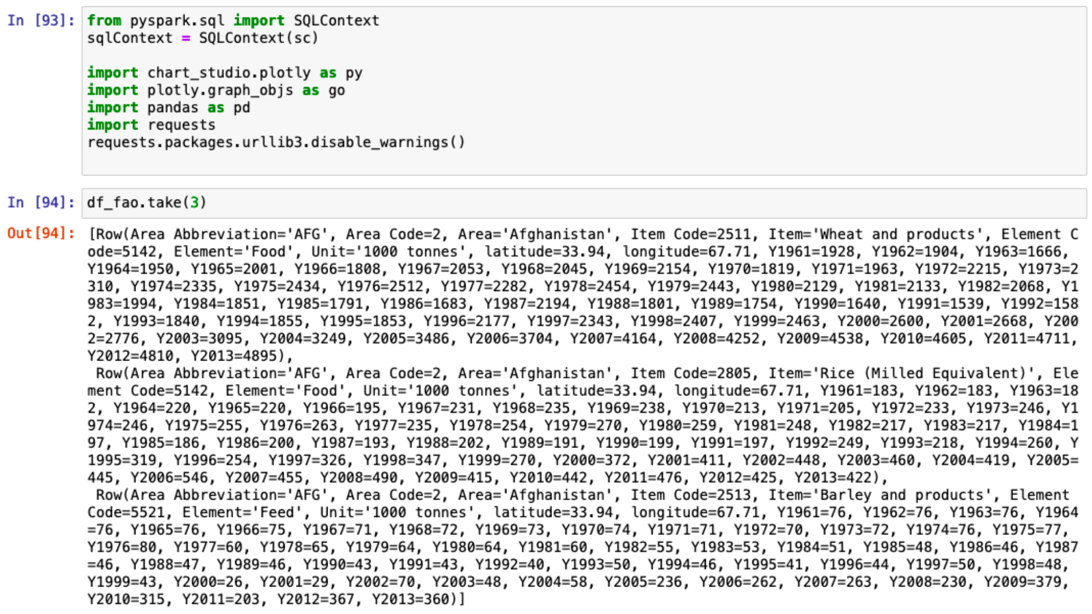
Importing the required libraries using below codes.



```
In [91]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns

In [92]: pip install chart_studio

         Requirement already satisfied: chart_studio in /opt/anaconda3/lib/python3.7/site-packages (1.1.0)
         Requirement already satisfied: six in /opt/anaconda3/lib/python3.7/site-packages (from chart_studio) (1.14.0)
         Requirement already satisfied: requests in /opt/anaconda3/lib/python3.7/site-packages (from chart_studio) (2.22.0)
         Requirement already satisfied: retrying>=1.3.3 in /opt/anaconda3/lib/python3.7/site-packages (from chart_studio)
         (1.3.3)
         Requirement already satisfied: plotly in /opt/anaconda3/lib/python3.7/site-packages (from chart_studio) (4.14.3)
         Requirement already satisfied: idna<2.9,>=2.5 in /opt/anaconda3/lib/python3.7/site-packages (from requests->chart_s
         tudio) (2.8)
         Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/anaconda3/lib/python3.7/site-package
         s (from requests->chart_studio) (1.25.8)
         Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.7/site-packages (from requests->cha
         rt_studio) (2019.11.28)
         Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/anaconda3/lib/python3.7/site-packages (from requests->
         chart_studio) (3.0.4)
         Note: you may need to restart the kernel to use updated packages.
```

Fig 1.29. EDA

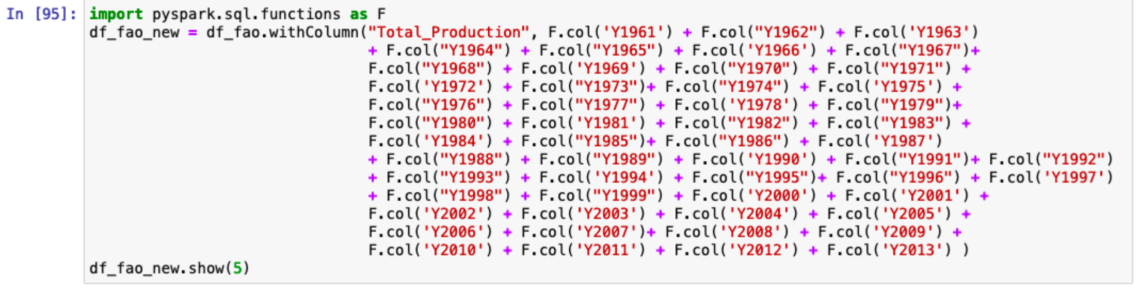
The below code shows the necessary libraries imported and take() function is used to display the columns.



```
In [93]: from pyspark.sql import SQLContext
         sqlContext = SQLContext(sc)

         import chart_studio.plotly as py
         import plotly.graph_objs as go
         import pandas as pd
         import requests
         requests.packages.urllib3.disable_warnings()

In [94]: df_fao.take(3)

Out[94]: [Row(Area Abbreviation='AFG', Area Code=2, Area='Afghanistan', Item Code=2511, Item='Wheat and products', Element C
         ode=5142, Element='Food', Unit='1000 tonnes', latitude=33.94, longitude=67.71, Y1961=1928, Y1962=1904, Y1963=1666,
         Y1964=1950, Y1965=2001, Y1966=1808, Y1967=2053, Y1968=2045, Y1969=2154, Y1970=1819, Y1971=1963, Y1972=2215, Y1973=2
         310, Y1974=2335, Y1975=2434, Y1976=2512, Y1977=2282, Y1978=2454, Y1979=2443, Y1980=2129, Y1981=2133, Y1982=2068, Y1
         983=1994, Y1984=1851, Y1985=1791, Y1986=1683, Y1987=2194, Y1988=1801, Y1989=1754, Y1990=1640, Y1991=1539, Y1992=158
         2, Y1993=1840, Y1994=1855, Y1995=1853, Y1996=2177, Y1997=2343, Y1998=2407, Y1999=2463, Y2000=2600, Y2001=2668, Y200
         2=2776, Y2003=3095, Y2004=3249, Y2005=3486, Y2006=3704, Y2007=4164, Y2008=4252, Y2009=4538, Y2010=4605, Y2011=4711,
         Y2012=4810, Y2013=4895),
          Row(Area Abbreviation='AFG', Area Code=2, Area='Afghanistan', Item Code=2805, Item='Rice (Milled Equivalent)', Ele
         ment Code=5142, Element='Food', Unit='1000 tonnes', latitude=33.94, longitude=67.71, Y1961=183, Y1962=183, Y1963=18
         2, Y1964=220, Y1965=220, Y1966=195, Y1967=231, Y1968=235, Y1969=238, Y1970=213, Y1971=205, Y1972=233, Y1973=246, Y1
         974=246, Y1975=255, Y1976=263, Y1977=235, Y1978=254, Y1979=270, Y1980=259, Y1981=248, Y1982=217, Y1983=217, Y1984=1
         97, Y1985=186, Y1986=200, Y1987=193, Y1988=202, Y1989=191, Y1990=199, Y1991=197, Y1992=249, Y1993=218, Y1994=260, Y
         1995=319, Y1996=254, Y1997=326, Y1998=347, Y1999=270, Y2000=372, Y2001=411, Y2002=448, Y2003=460, Y2004=419, Y2005=
         445, Y2006=546, Y2007=455, Y2008=490, Y2009=415, Y2010=442, Y2011=476, Y2012=425, Y2013=422),
          Row(Area Abbreviation='AFG', Area Code=2, Area='Afghanistan', Item Code=2513, Item='Barley and products', Element
         Code=5521, Element='Feed', Unit='1000 tonnes', latitude=33.94, longitude=67.71, Y1961=76, Y1962=76, Y1963=76, Y1964
         =76, Y1965=76, Y1966=75, Y1967=71, Y1968=72, Y1969=73, Y1970=74, Y1971=71, Y1972=70, Y1973=72, Y1974=76, Y1975=77,
         Y1976=80, Y1977=60, Y1978=65, Y1979=64, Y1980=64, Y1981=60, Y1982=55, Y1983=53, Y1984=51, Y1985=48, Y1986=46, Y1987
         =46, Y1988=47, Y1989=46, Y1990=43, Y1991=43, Y1992=40, Y1993=50, Y1994=46, Y1995=41, Y1996=44, Y1997=50, Y1998=48,
         Y1999=43, Y2000=26, Y2001=29, Y2002=70, Y2003=48, Y2004=58, Y2005=236, Y2006=262, Y2007=263, Y2008=230, Y2009=379,
         Y2010=315, Y2011=203, Y2012=367, Y2013=360)]
```

Fig 1.30. Take() function

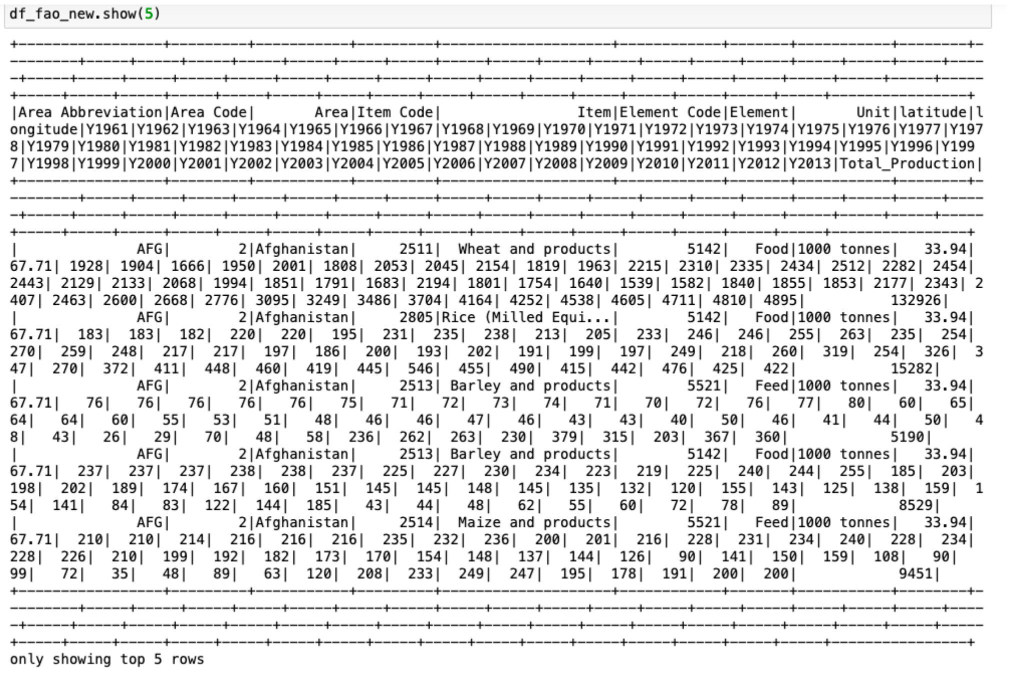
Creating a new column named ‘Total_Production’ by adding all the values from 1961 to 2013.



```
In [95]: import pyspark.sql.functions as F
         df_fao_new = df_fao.withColumn("Total_Production", F.col('Y1961') + F.col("Y1962") + F.col('Y1963')
                                        + F.col("Y1964") + F.col("Y1965") + F.col('Y1966') + F.col("Y1967")+
                                        F.col("Y1968") + F.col('Y1969') + F.col("Y1970") + F.col("Y1971") +
                                        F.col('Y1972') + F.col("Y1973")+ F.col("Y1974") + F.col('Y1975') +
                                        F.col("Y1976") + F.col("Y1977") + F.col('Y1978') + F.col("Y1979")+
                                        F.col("Y1980") + F.col('Y1981') + F.col("Y1982") + F.col("Y1983") +
                                        F.col('Y1984') + F.col("Y1985")+ F.col("Y1986") + F.col('Y1987')
                                        + F.col("Y1988") + F.col("Y1989") + F.col('Y1990') + F.col("Y1991")+ F.col("Y1992")
                                        + F.col("Y1993") + F.col('Y1994') + F.col("Y1995")+ F.col("Y1996") + F.col('Y1997')
                                        + F.col("Y1998") + F.col("Y1999") + F.col('Y2000') + F.col('Y2001') +
                                        F.col('Y2002') + F.col('Y2003') + F.col('Y2004') + F.col('Y2005') +
                                        F.col('Y2006') + F.col('Y2007')+ F.col('Y2008') + F.col('Y2009') +
                                        F.col('Y2010') + F.col('Y2011') + F.col('Y2012') + F.col('Y2013') )
         df_fao_new.show(5)
```

Fig 1.31. withColumn()

The result for the above code is shown below:

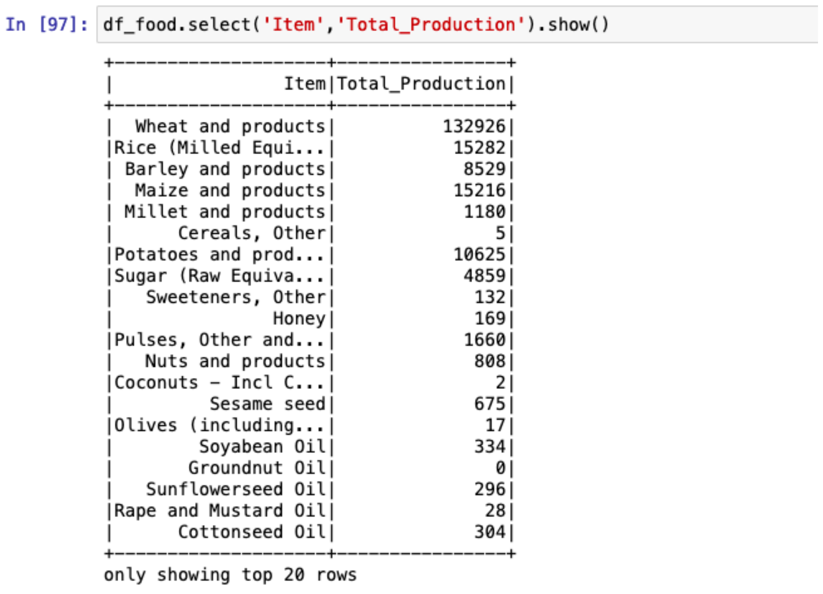


```
df_fao_new.show(5)
```

```
+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----------------+
|Area Abbreviation|Area Code|       Area|Item Code|                Item|Element Code|Element|       Unit|latitude|l
ongitude|Y1961|Y1962|Y1963|Y1964|Y1965|Y1966|Y1967|Y1968|Y1969|Y1970|Y1971|Y1972|Y1973|Y1974|Y1975|Y1976|Y1977|Y197
8|Y1979|Y1980|Y1981|Y1982|Y1983|Y1984|Y1985|Y1986|Y1987|Y1988|Y1989|Y1990|Y1991|Y1992|Y1993|Y1994|Y1995|Y1996|Y199
7|Y1998|Y1999|Y2000|Y2001|Y2002|Y2003|Y2004|Y2005|Y2006|Y2007|Y2008|Y2009|Y2010|Y2011|Y2012|Y2013|Total_Production|
+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----------------+
|              AFG|        2|Afghanistan|     2511|  Wheat and products|        5142|   Food|1000 tonnes|   33.94|
67.71| 1928| 1904| 1666| 1950| 2001| 1808| 2053| 2045| 2154| 1819| 1963| 2215| 2310| 2335| 2434| 2512| 2282| 2454|
2443| 2129| 2133| 2068| 1994| 1851| 1791| 1683| 2194| 1801| 1754| 1640| 1539| 1582| 1840| 1855| 1853| 2177| 2343| 2
407| 2463| 2600| 2668| 2776| 3095| 3249| 3486| 3704| 4164| 4252| 4538| 4605| 4711| 4810| 4895|          132926|
|              AFG|        2|Afghanistan|     2805|Rice (Milled Equi...|        5142|   Food|1000 tonnes|   33.94|
67.71|  183|  183|  182|  220|  220|  195|  231|  235|  238|  213|  205|  233|  246|  246|  255|  263|  235|  254|
270|  259|  248|  217|  217|  197|  186|  200|  193|  202|  191|  199|  197|  249|  218|  260|  319|  254|  326|  3
47|  270|  372|  411|  448|  460|  419|  445|  546|  455|  490|  415|  442|  476|  425|  422|           15282|
|              AFG|        2|Afghanistan|     2513| Barley and products|        5521|   Feed|1000 tonnes|   33.94|
67.71|   76|   76|   76|   76|   76|   75|   71|   72|   73|   74|   71|   70|   72|   76|   77|   80|   60|   65|
64|   64|   60|   55|   53|   51|   48|   46|   46|   47|   46|   43|   43|   40|   50|   46|   41|   44|   50|   4
8|   43|   26|   29|   70|   48|   58|  236|  262|  263|  230|  379|  315|  203|  367|  360|            5190|
|              AFG|        2|Afghanistan|     2513| Barley and products|        5142|   Food|1000 tonnes|   33.94|
67.71|  237|  237|  237|  238|  238|  237|  225|  227|  230|  234|  223|  219|  225|  240|  244|  255|  185|  203|
198|  202|  189|  174|  167|  160|  151|  145|  145|  148|  145|  135|  132|  120|  155|  143|  125|  138|  159|  1
54|  141|   84|   83|  122|  144|  185|   43|   44|   48|   62|   55|   60|   72|   78|   89|            8529|
|              AFG|        2|Afghanistan|     2514|  Maize and products|        5521|   Feed|1000 tonnes|   33.94|
67.71|  210|  210|  214|  216|  216|  216|  235|  232|  236|  200|  201|  216|  228|  231|  234|  240|  228|  234|
228|  226|  210|  199|  192|  182|  173|  170|  154|  148|  137|  144|  126|   90|  141|  150|  159|  108|   90|
99|   72|   35|   48|   89|   63|  120|  208|  233|  249|  247|  195|  178|  191|  200|  200|            9451|
+-----------------+---------+-----------+---------+--------------------+------------+-------+-----------+--------+-
--------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----------------+
only showing top 5 rows
```

Fig 1.32. Show() function

Code using select() function to display total production of the items is shown below.

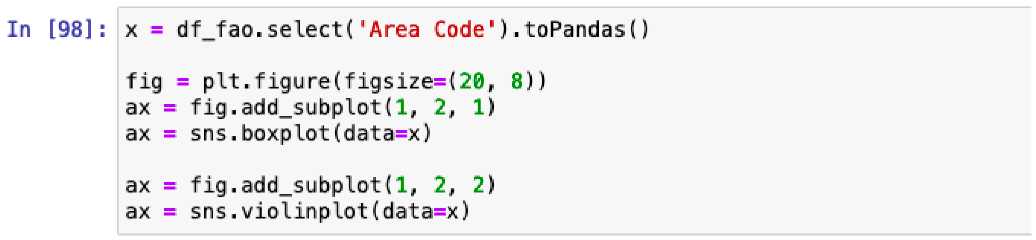


```
In [97]: df_food.select('Item','Total_Production').show()
```

```
+--------------------+----------------+
|                Item|Total_Production|
+--------------------+----------------+
|  Wheat and products|          132926|
|Rice (Milled Equi...|           15282|
| Barley and products|            8529|
|  Maize and products|           15216|
| Millet and products|            1180|
|      Cereals, Other|               5|
|Potatoes and prod...|           10625|
|Sugar (Raw Equiva...|            4859|
|   Sweeteners, Other|             132|
|               Honey|             169|
|Pulses, Other and...|            1660|
|   Nuts and products|             808|
|Coconuts - Incl C...|               2|
|         Sesame seed|             675|
|Olives (including...|              17|
|         Soyabean Oil|             334|
|        Groundnut Oil|               0|
|    Sunflowerseed Oil|             296|
|Rape and Mustard Oil|              28|
|       Cottonseed Oil|             304|
+--------------------+----------------+
only showing top 20 rows
```

Fig 1.33. Select() function

2.8 Box Plot and Violin Plots



```
In [98]: x = df_fao.select('Area Code').toPandas()

         fig = plt.figure(figsize=(20, 8))
         ax = fig.add_subplot(1, 2, 1)
         ax = sns.boxplot(data=x)

         ax = fig.add_subplot(1, 2, 2)
         ax = sns.violinplot(data=x)
```

Fig 1.34. Code for Box and Violin Plots

Results:

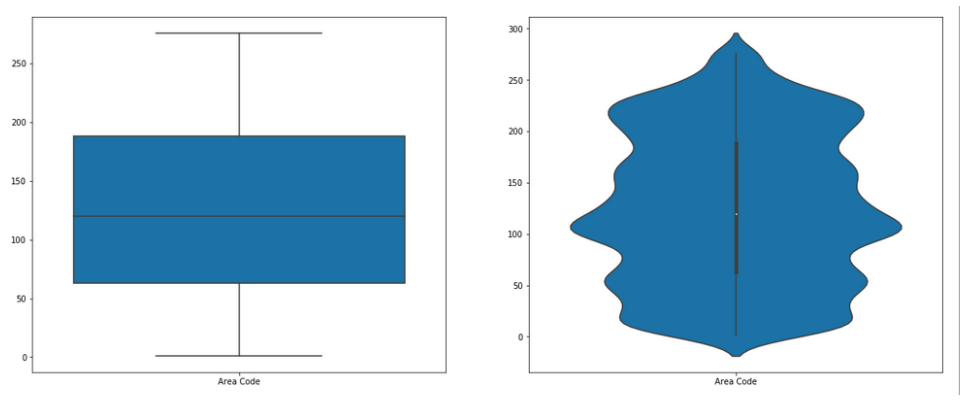


Fig 1.35. Box and Violin plots for Area code

In [108]:
```
x = df_fao.select('Y2013').toPandas()

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)

ax = fig.add_subplot(1, 2, 2)
ax = sns.violinplot(data=x)
```

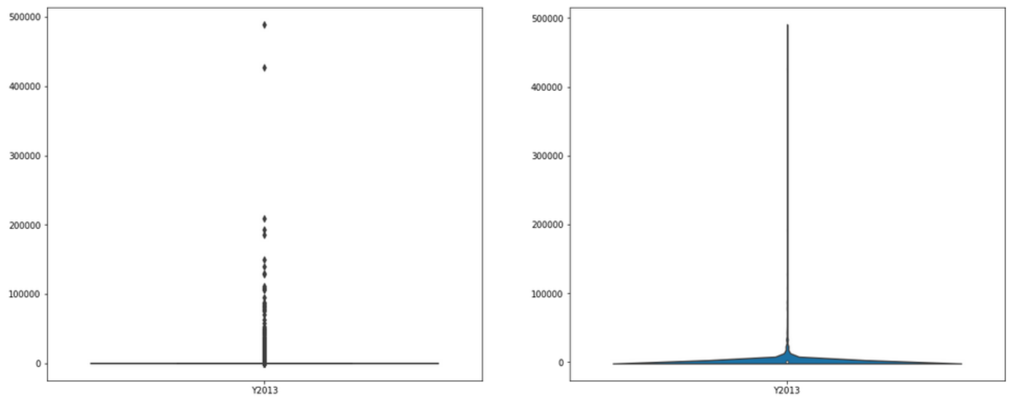


Fig 1.36. Box and Violin plots for Y2013

In [107]:
```
x = df_fao.select('Y2011').toPandas()

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)

ax = fig.add_subplot(1, 2, 2)
ax = sns.violinplot(data=x)
```

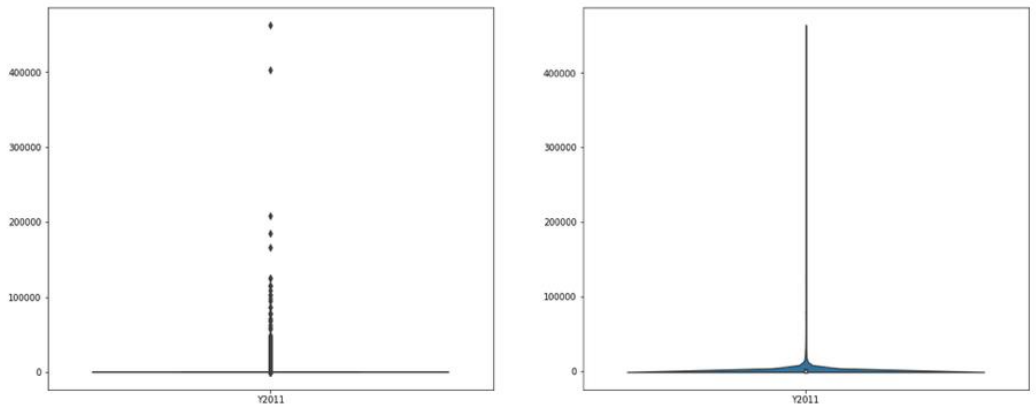


Fig 1.37. Box and Violin plots for Y2011

```
In [106]: x = df_fao.select('Y2001').toPandas()

          fig = plt.figure(figsize=(20, 8))
          ax = fig.add_subplot(1, 2, 1)
          ax = sns.boxplot(data=x)

          ax = fig.add_subplot(1, 2, 2)
          ax = sns.violinplot(data=x)
```

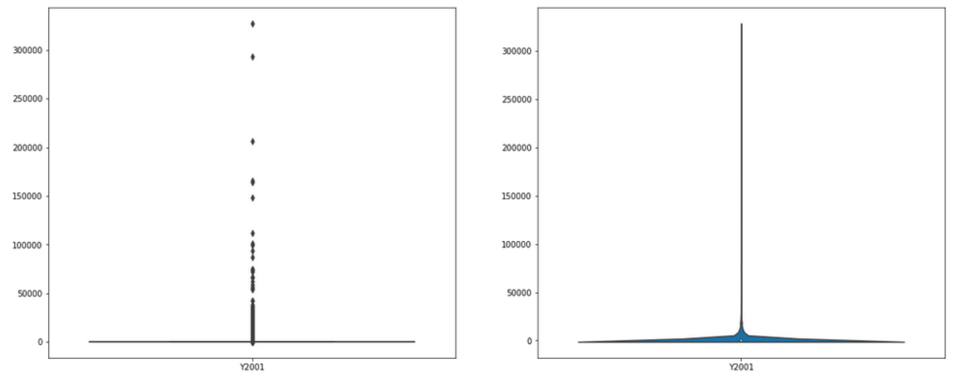


Fig 1.38. Box and Violin plots for 2011

```
In [105]: x = df_fao.select('Y1991').toPandas()

          fig = plt.figure(figsize=(20, 8))
          ax = fig.add_subplot(1, 2, 1)
          ax = sns.boxplot(data=x)

          ax = fig.add_subplot(1, 2, 2)
          ax = sns.violinplot(data=x)
```

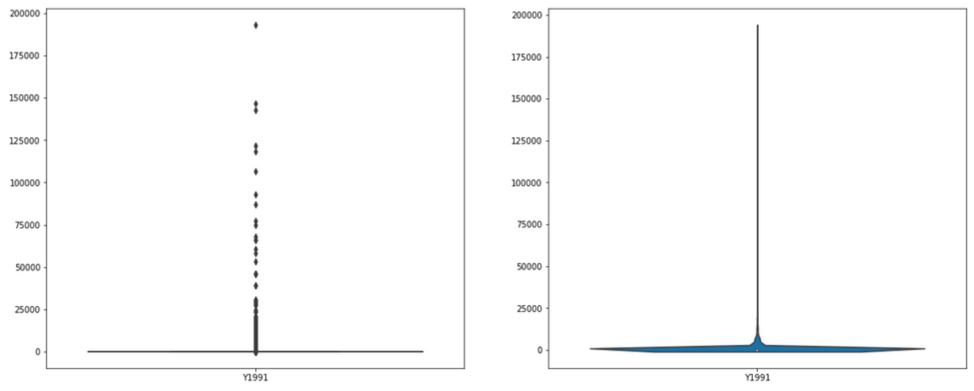


Fig 1.39. Box and Violin plots for y1991

```
In [103]: x = df_fao.select('Y1971').toPandas()

          fig = plt.figure(figsize=(20, 8))
          ax = fig.add_subplot(1, 2, 1)
          ax = sns.boxplot(data=x)

          ax = fig.add_subplot(1, 2, 2)
          ax = sns.violinplot(data=x)
```

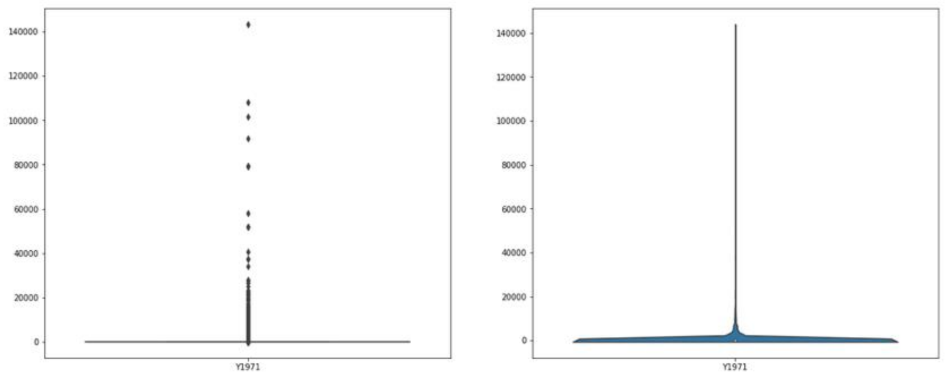
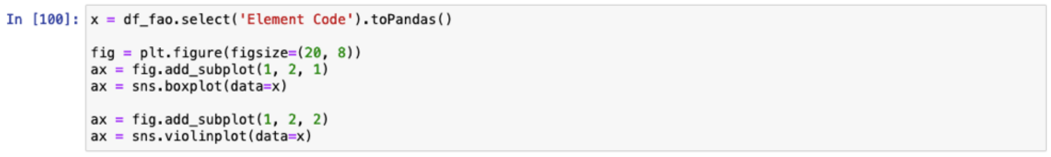


Fig 1.40. Box and Violin plots for Y1971



```
In [100]: x = df_fao.select('Element Code').toPandas()

          fig = plt.figure(figsize=(20, 8))
          ax = fig.add_subplot(1, 2, 1)
          ax = sns.boxplot(data=x)

          ax = fig.add_subplot(1, 2, 2)
          ax = sns.violinplot(data=x)
```

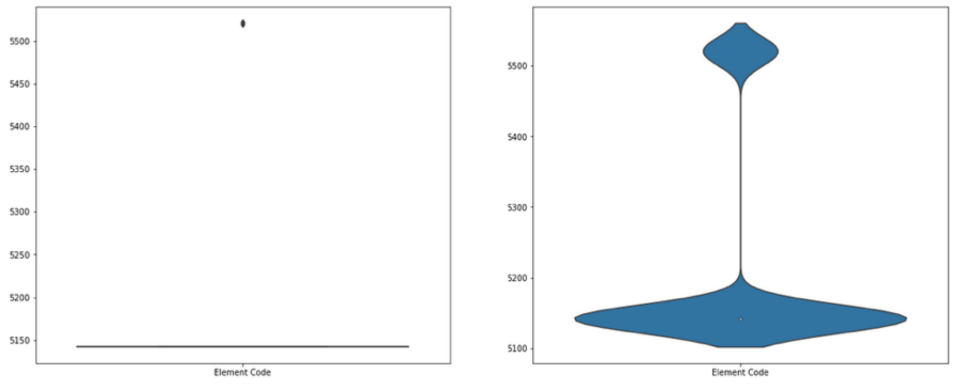


Fig 1.41. Box and Violin plots for Element code

2.9 K-means Clustering Implementation:

Machine learning has an adequate number of algorithms that can solve real-life problems easily. It is a concept where the machine takes its previous data as a feedback/input and predict or classify the output. Many big companies such as Netflix, Amazon, Google are using these machine learning techniques to improve their customer experience. Basically, these algorithms are divided into two types one is supervised and the other is unsupervised learning. Some of the machine learning algorithms are mentioned below.

1. **Supervised Learning:**
   It is called supervised learning because the user already knows the output and will be supervising/monitoring the model to get correct output. It is further divided into classification and regression type.

2. **Unsupervised Learning:**
   Unlike supervised learning models, these algorithms have no desired results and there's no supervision required as they are left to their own performance. It is further divided into clustering and association techniques.

Some of the widely used machine learning algorithms are mentioned below:

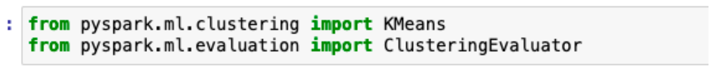
a. Linear Regression
b. Decision Tree Classifier
c. Logistic Regression
d. Random Forest algorithm
e. K Nearest Neighbour
f. K-means Clustering.

We are going to use one K Means Clustering algorithm in our project to solve the FAO dataset problem.

K-means Clustering:

It is the most popular algorithm among the clustering techniques. Clustering is the process where data gets divided into clusters based on the similarities. The main aim of this algorithm is to gain insights from the data by grouping it depending on similar features. Here, K in K-means denotes the fixed number of clusters used in solving the problem. Below code shows the required libraries to be imported for K-means algorithm:

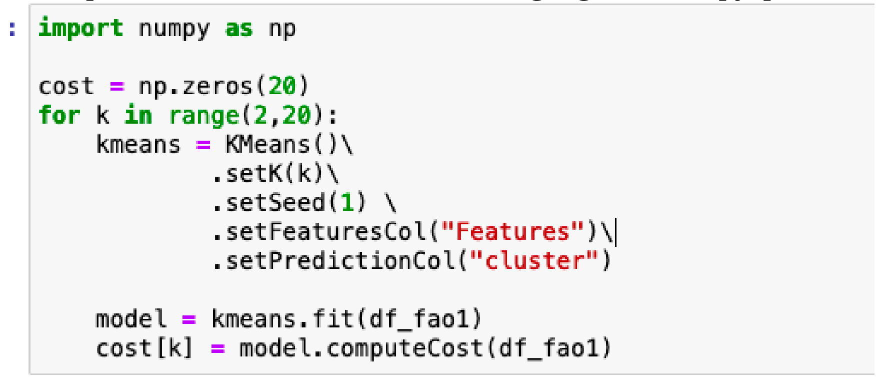
K Means Clustering



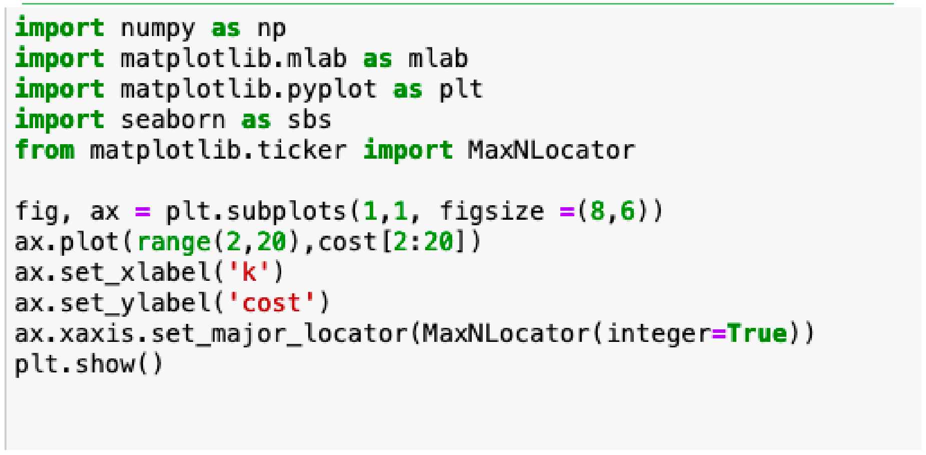
```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

Fig 1.42. Libraries for K-means Clustering

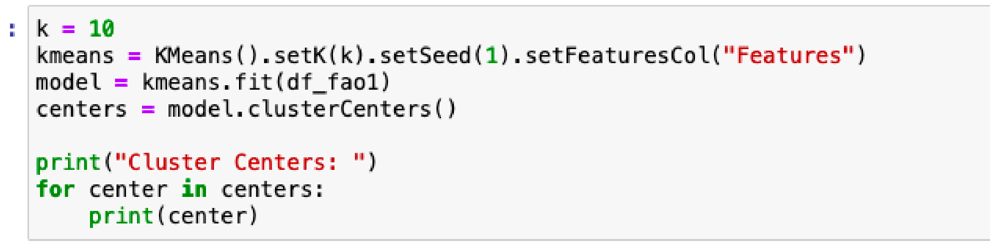
Below code shows the implementation of K-means clustering algorithm in pyspark.



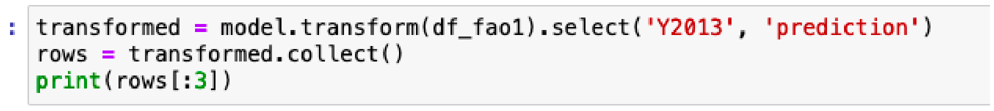
```
import numpy as np

cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans()\
            .setK(k)\
            .setSeed(1) \
            .setFeaturesCol("Features")\
            .setPredictionCol("cluster")

    model = kmeans.fit(df_fao1)
    cost[k] = model.computeCost(df_fao1)
```



```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sbs
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```



```
k = 10
kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("Features")
model = kmeans.fit(df_fao1)
centers = model.clusterCenters()

print("Cluster Centers: ")
for center in centers:
    print(center)
```



```
transformed = model.transform(df_fao1).select('Y2013', 'prediction')
rows = transformed.collect()
print(rows[:3])
```

Fig 1.43. K-means algorithm

2.10 Visualisations using Power BI:

Power BI is an analytics software provided my Microsoft which allows interactive usage for end user to deal and explore the data. In this project, I have used Power BI for visualising the data.
Some of the visualisations are as mentioned below:

The figure below represents total food and feed produced all over the world from 2000-2013.

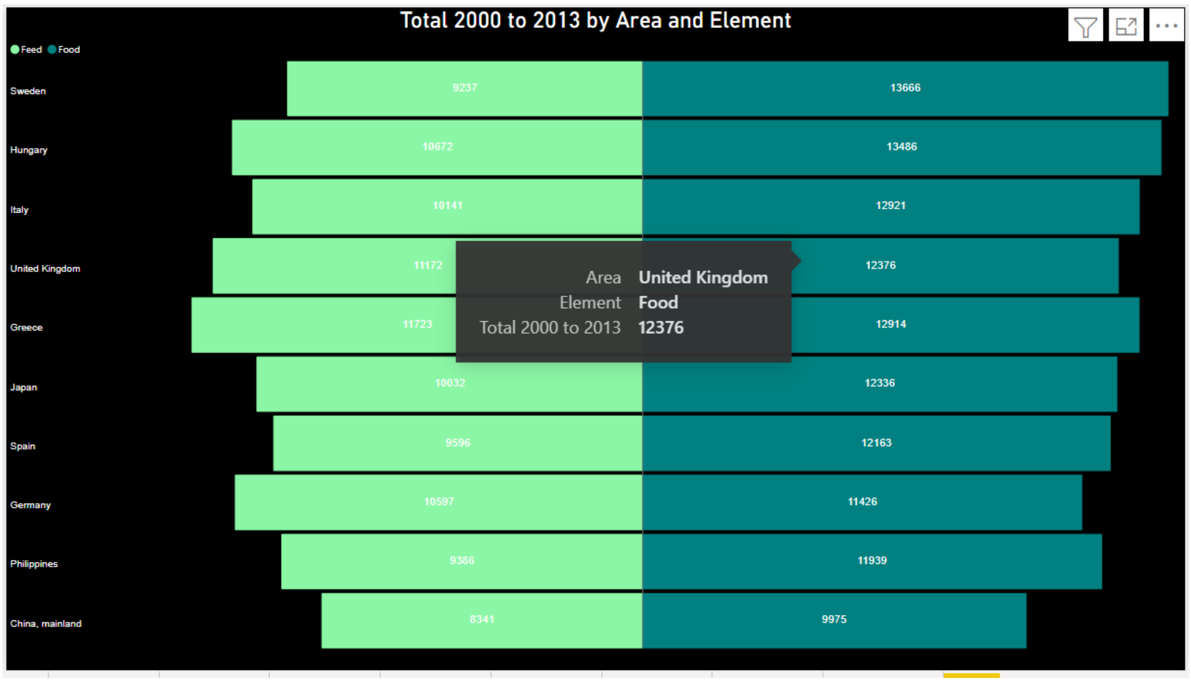


Fig 1.44. Amount of Food and Feed for top 10 countries

Similarly, the below figure shows the feed production all over the world from 2000 to 2013.

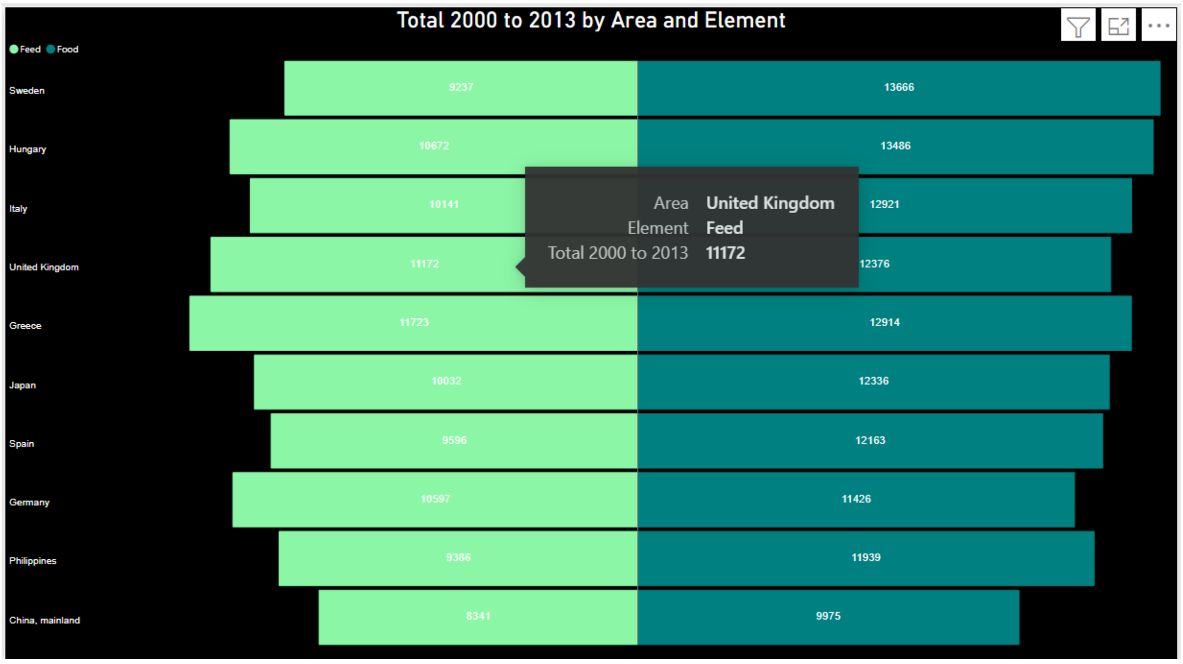


Fig 1.45. Amount of Food and Feed for top 10 countries

The below chart represents a bar graph which shows the food distribution for the top 10 countries.

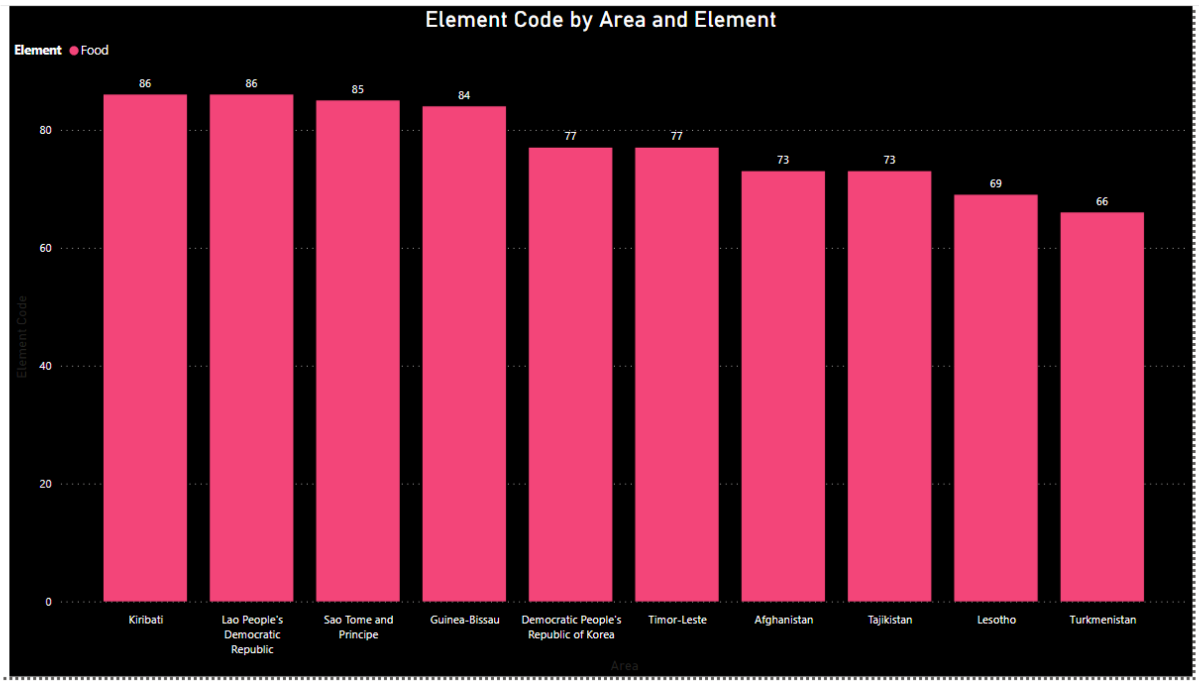


Fig 1.46. Food distribution by Area

I used a pie chart for showing the feed and food percentage in the data as shown in the below figure. We can clearly see that the percentage of food produced is much higher than the feed produced.

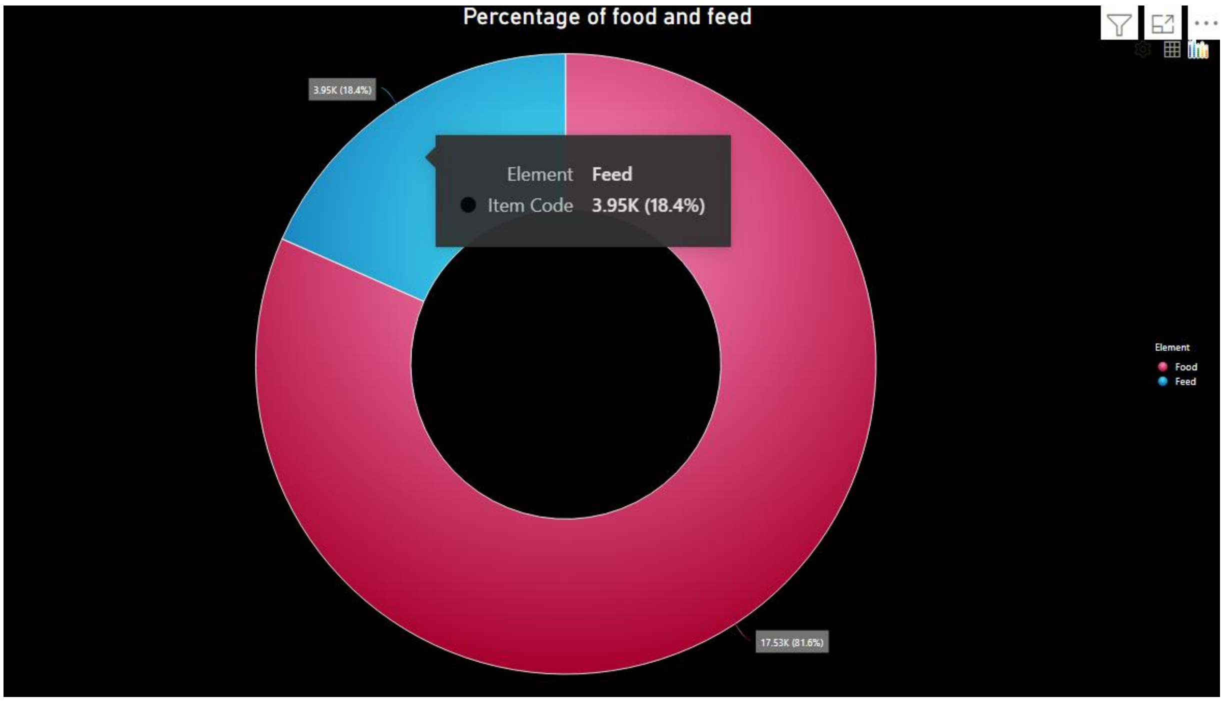


Fig 1.47. Food and Feed Percentage

Below chart is a representation of Donut chart which shows the distribution of Items in the data. Items such as milk, cereals, fish, eggs, have less percentage compared to the combination of all other items.

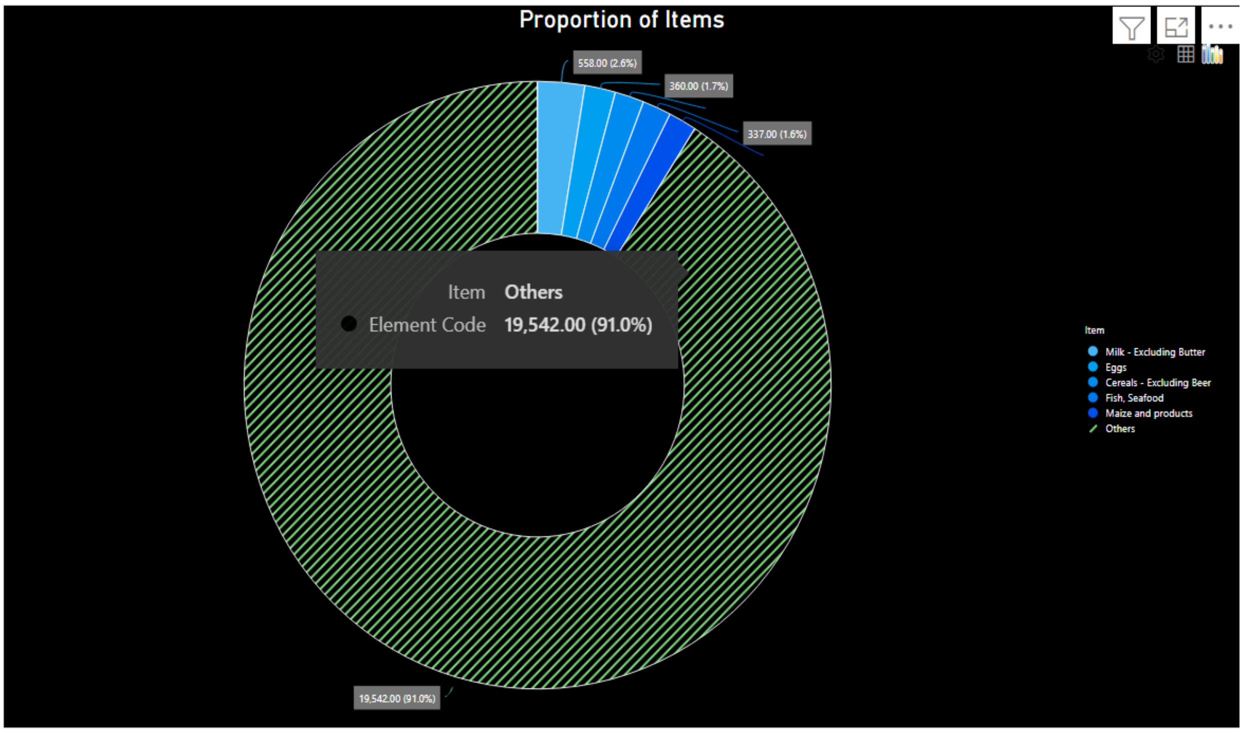


Fig 1.48. Item Percentage

The below figure shows a world map representation of all the areas available in the dataset. The blue dots represent the countries. For example, you can see in the map that represents country India and its production.

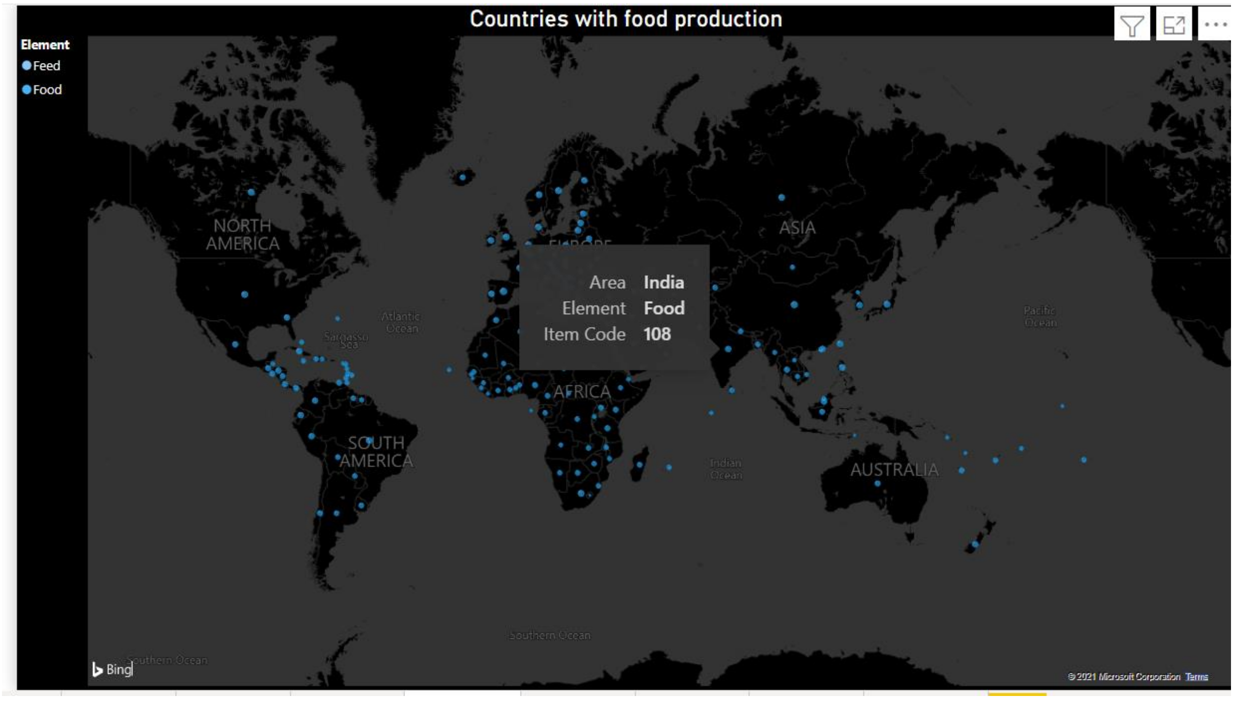


Fig 1.49. Map showing different countries

Below chart represents the food produced in the year 1961 by top 10 countries. We can see that USA followed by China are in the leading positions.

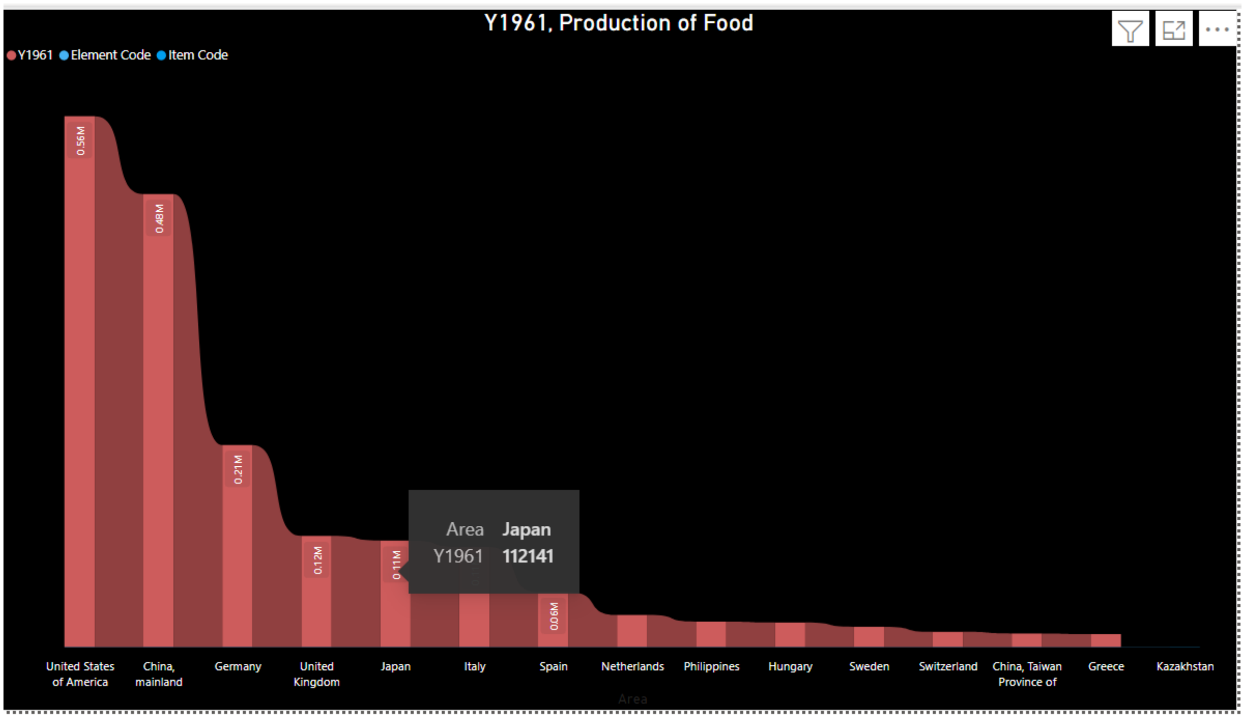


Fig 1.50. Production of Food in year 1961

In the figure shown below, the production of food by top 10 countries in the year 2013 is represented. By this we can see that China leads the race followed by United States Of America.

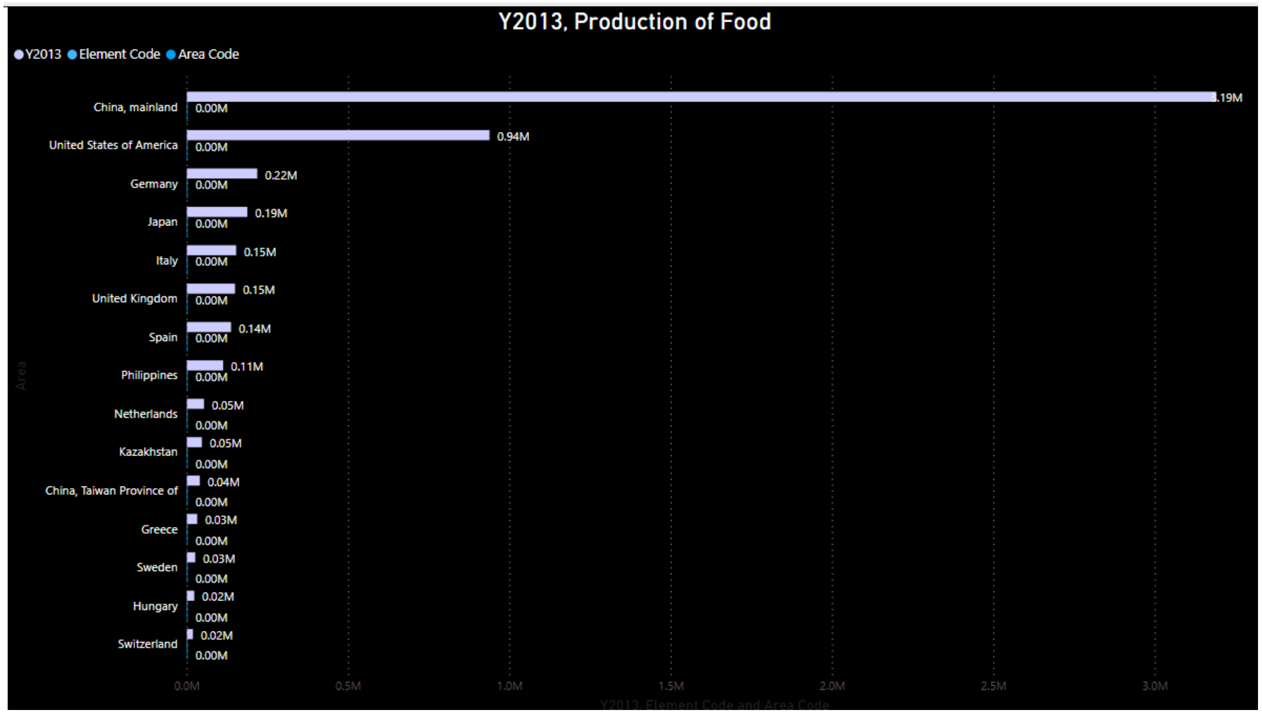


Fig 1.51. Production of food in Y2013

Lastly, I created a dashboard using Power BI as shown below. The below dashboard represents different charts used to visualise the data.

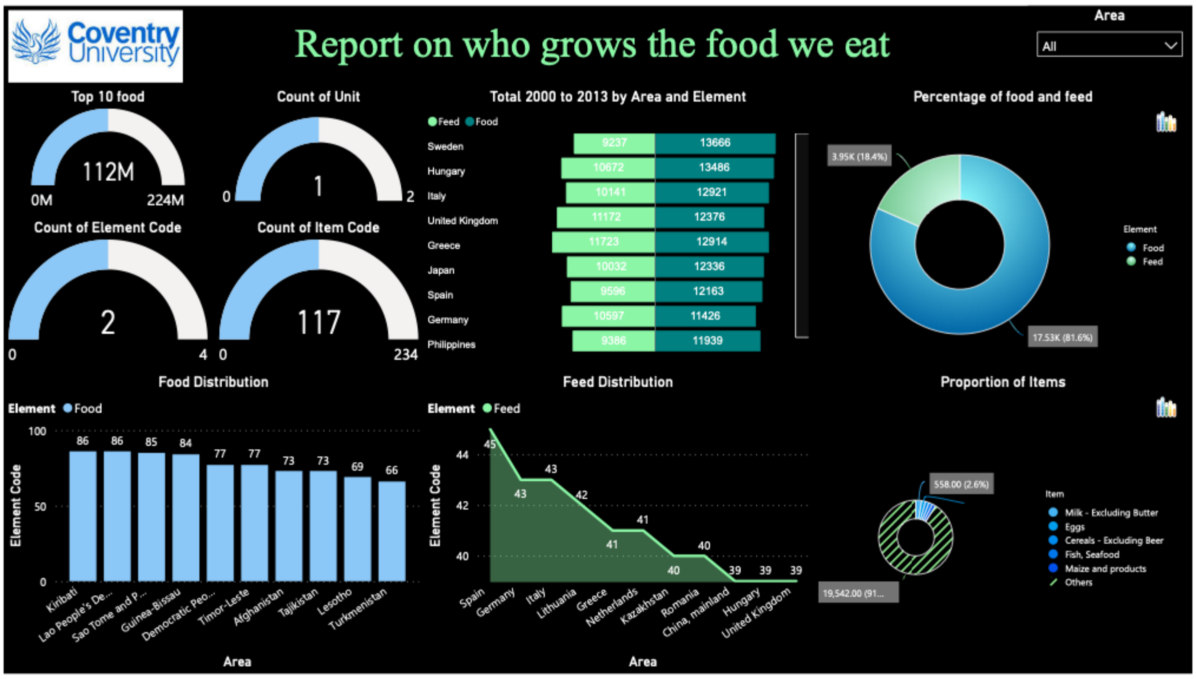
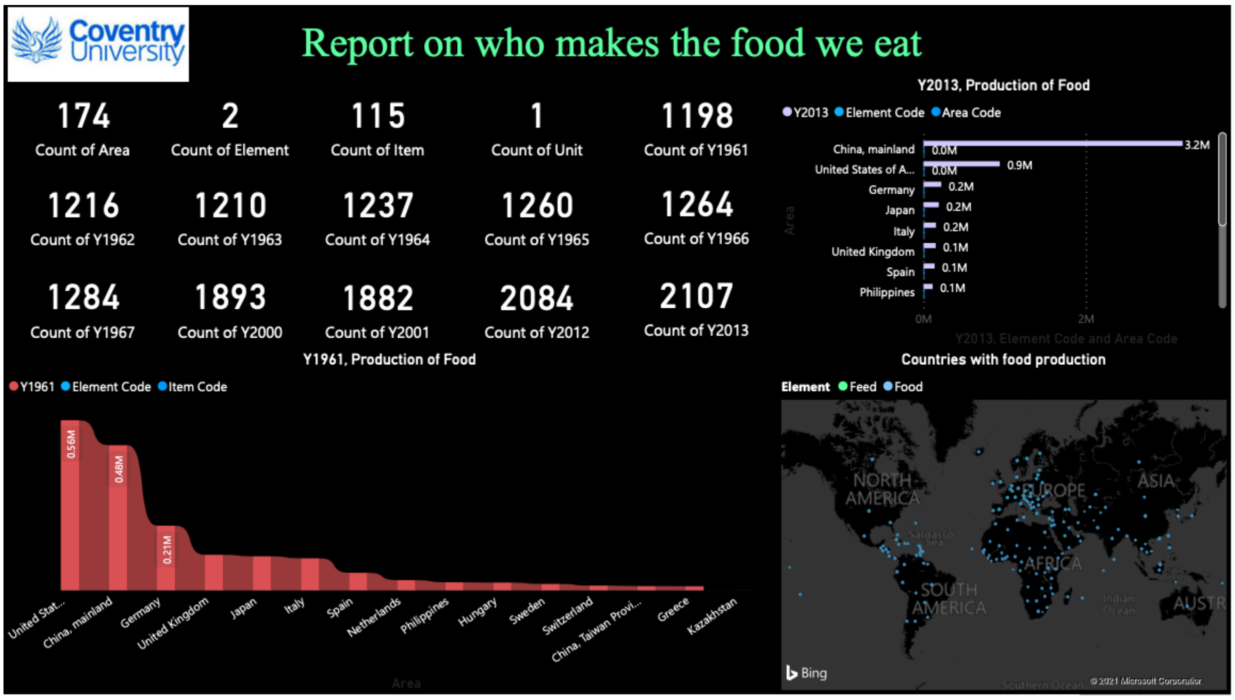


Fig 1.52. Dashboard showing multiple charts

The below figure is a dashboard representation of different charts used and it also represents the count of different columns in



174	2	115	1	1198
Count of Area	Count of Element	Count of Item	Count of Unit	Count of Y1961
1216	1210	1237	1260	1264
Count of Y1962	Count of Y1963	Count of Y1964	Count of Y1965	Count of Y1966
1284	1893	1882	2084	2107
Count of Y1967	Count of Y2000	Count of Y2001	Count of Y2012	Count of Y2013

Fig 1.53. Dashboard showing multiple charts

Conclusion:

In today’s world, food production and its transportation has become very crucial as population is increasing day-to-day. So, every country needs to focus on its food production and research techniques to make sure that no person is left unfed. Not only food, but also feed production is important to maintain the food chain balance. As we have seen in the analysis, there is a variation between many countries’ food and feed production, so it’s very important to maintain good food production so that transporting food to other countries can be limited, hence reducing the transportation costs.

With the development and implementation of technologies like machine learning and data science, it is becoming very easy for researchers and developers to solve complex problems. So we have used PySpark and machine learning in this project to analyse the food data. We can see from the analysis that countries like China and United States of America are consistently being on top in food production.

Appendix:

Dataset Link: https://www.kaggle.com/dorbicycle/world-foodfeed-production

GitHub Link for code: https://github.com/Rohith655/Data_Science/blob/main/FAO.ipynb

References:

1. https://spark.apache.org/docs/latest/index.html#where-to-go-from-here.
2. J. Fu, J. Sun and K. Wang, "Spark-a big data processing platform for machine learning", *Industrial Informatics-Computing Technology Intelligent Technology Industrial Information Integration (ICIICII) 2016 International Conference on*, pp. 48-51, 2016.
3. Yicheng Huang, Xingtu Lan, Xing Chen and Wenzhong Guo, "Towards Model Based Approach to Hadoop Deployment and Configuration", *2015 12th Web Information System and Application Conference (WISA)*, pp. 79-84, 2015.
4. "Spark notes for beginners & experienced", *Website*, 6 2016, [online] Available: https://data-flair.training/blogs/spark-notes/
5. E. Evans, D. Sauvant, P. Ud&eacute;n, “Mathematical Models that Predict the Effects of Feed Characteristics on Animal Performance”May 2008|Volume 143, Issues 1-4.