Life Expectancy (WHO)

(7153CEM)

Coursework

Big Data Analytics and Visualization Using PySpark

Student Name: Aleena Alby

Student ID: 11865340

1. Introduction

Machine learning involves the design and study of algorithms that can make predictions and learn from data. An analysis of big data is the focus of this project, which identifies and selects suitable analytical techniques and discusses the results. The project entangles applications of Apache Spark. Python is used for this study. Pyspark is a combination of python and spark and also it supports a lot of libraries useful for this project. This project structured Pyspark and Jupyter Notebook to construct a machine learning model.

The dataset used for this project is downloaded from Kaggle. The dataset name is **" Life expectancy(WHO)"**. The dataset contains values of 193 countries collected from the WHO repository. This project considers 15 years of data from 2000 to 2015. To keep authentic and coherent data external dataset has been merged into a single dataset. The final merged dataset contains 23 columns and 2938 rows. Missing values and data duplication is handled during the data preprocessing stage. The data then feeds into Tableau to visualize the dataset.

In nutshell, the study is concerned with the life expectancy and factors affecting it, as well as how countries with developed or developing status affect a country's GDP and total expenditure.

2 . BACKGROUND AND IMPLEMENTATION

2.1 Background Study

2.1.1. Spark

Apache Spark is a software application for processing data in a distributed environment. It can handle a large amount of data. Python, R, Scala, and Java are supported by SPARK's Application Programming Interface (API). Additionally, Spark includes higher-level libraries for SQL queries, graph processing and machine learning. Python was chosen for this study because of the libraries, and Pyspark was selected since it is a combination of Python and Spark.

2.1.2. Pyspark

PySpark is a Spark library written in Python that can run Python applications using Apache Spark capabilities. The Apache Spark library is written in Scala. To support Spark with python new tool is developed which is called Pyspark. PySpark applications run 100x faster than traditional systems. Most data scientists and analytics experts today use Python because of its rich library set. Integrating Python and Spark is a great benefit for them.

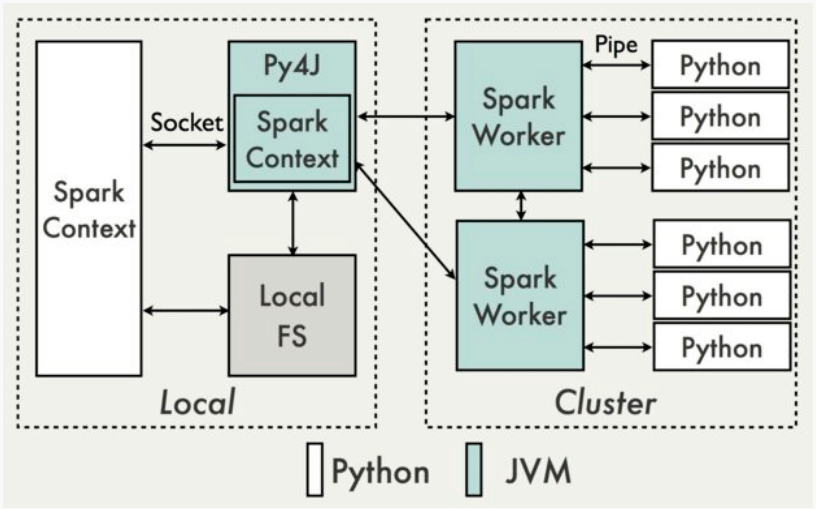


Fig (2.1.2) Illustration of PySpark data flow.

2.1.3 Hadoop

Hadoop is an Apache open-source framework written in Java that is used to process large datasets. A Hadoop framework application runs in an environment that enables distributed storage and computation across a cluster of computers. It can be scaled from a single machine to thousands, with each offering local computation and storage.

2.1.4 Tableau

Tableau is an interactive data visualization tool that allows you to generate interactive and appropriate visualizations in the form of dashboards and spreadsheets to gain business insights for better business development. As well as advanced visualization, quick analytics, easy sharing, and interactive presentations, it can be used for many different purposes.

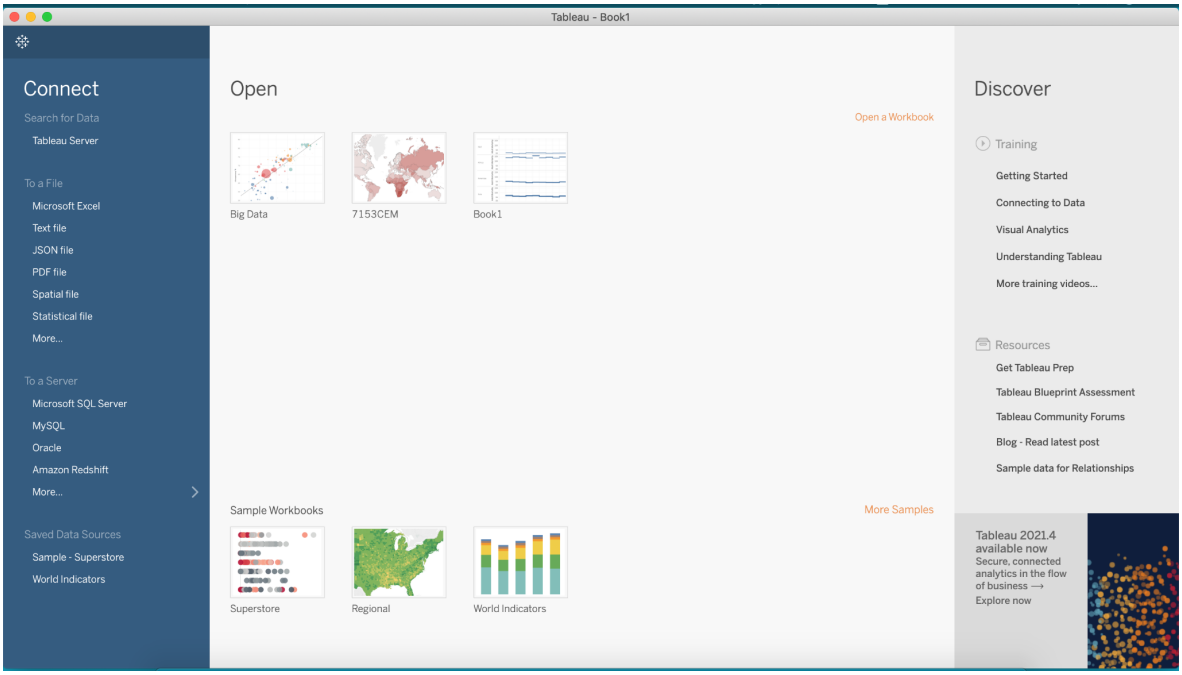
2.1.5 Jupyter notebook

Using Jupyter Notebook, you can create and share documents with live code, equations, visuals, and text. Cleaning and transforming data, statistical modelling, data visualization, machine learning, and many other applications are all possible.

2.2 Software Installation

2.2.1 Tableau

Tableau Software is a tool that allows people to see and understand the data. This software can be downloaded from the official Tableau website.
Then Install and run



Fig(2.2.1) Interface of Tableau

2.2.3 Pyspark

Pyspark Installation code is given below

```
Alans-MacBook-Pro:~ aleenaalby$ pip3 install pyspark
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 32 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 26.1 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=281853646 sha256=ddca0e2f96a1a604e6c445eaca1e99687b7c26caf12562dd8c2ba62fea26ac4
2
  Stored in directory: /Users/aleenaalby/Library/Caches/pip/wheels/52/45/50/69db7b6e1da74a1b9fcc097827db9185cb8627117de852731e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

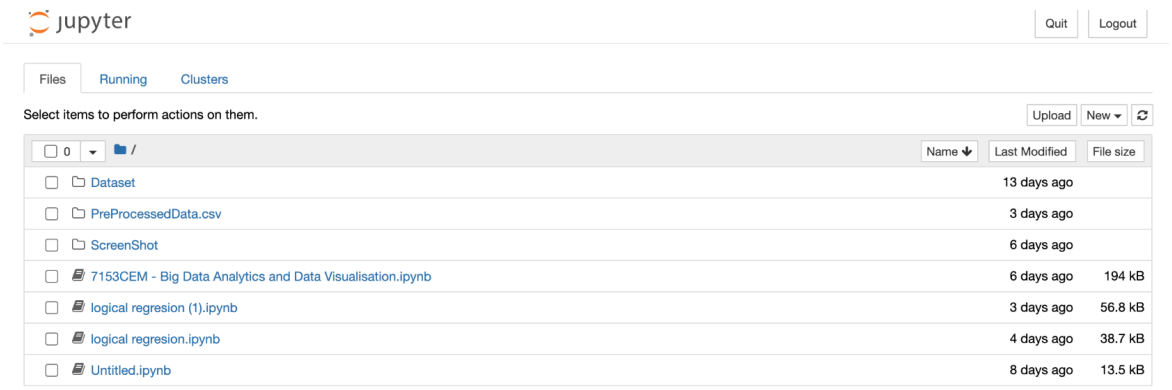
Fig(2.2.3)

2.2.4 Jupyter notebook

```
aleenaalby$ PYSPARK_DRIVER_PYTHON="jupyter"
aleenaalby$ export PYSPARK_DRIVER_PYTHON="jupyter"
aleenaalby$ export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
aleenaalby$  jupyter notebook
```

Fig(2.2.4.1)

This is how the jupyter notebook will look after installation:



Fig(2.2.4.2)

2.3. The Dataset

The dataset selected for this project is **Life Expectancy (WHO)**. The Global Health Observatory (GHO) data repository under World Health Organization (WHO) keeps track of the health status as well as many other related factors for all countries. For the purposes of health data analysis, the datasets are made public. In this dataset, there are data from the years 2000-2015 for 193 countries. This dataset consists of 22 Columns and 2938 rows. To keep authentic and coherent data external dataset has been merged into a single dataset. The final merged dataset contains 23 columns and 2938 rows. An explanation of each attribute is provided below.

Country: The name of the country for which the rest of the data is provided.

Year: Data from the year from 2000 to 2015.

Status: Country's development status (Developed of Developing)

Life Expectancy: Number of years that a person can expect to live.

Adult Mortality: The death rate of adults per thousand people in a year both men and women.

Infant Deaths: The death rate of infants per thousands population in a year.

Alcohol: Consumption of alcohol per capita for those over 15 years old (litres).

Percentage Expenditure: The percentage of GDP that is spent on health.

Hepatitis B: Immunization coverage for Hepatitis B (HepB).

Measles: Number of reported cases of Measles

BMI: The average body mass index of the population.

Under-five deaths: The number of under-five deaths.

Polio: Number of reported cases of Polio.

Total Expenditure: Total expense made by the country.

Diphtheria: Immunization coverage for diphtheria.

HIV/AIDS: Number of HIV/AIDS death.

GDP: Gross Domestic Product in that particular year.

Population: Population of the country in a particular year.

Thinness (1-19 years): Average thinness of individual in a particular year

Thinness (5-9 years): Average thinness of individual in a particular year

Income Composition of Resources: The composition of income from resources

Continent: Continent of each nation.

Schooling: Average education years.

2.4 Implementation

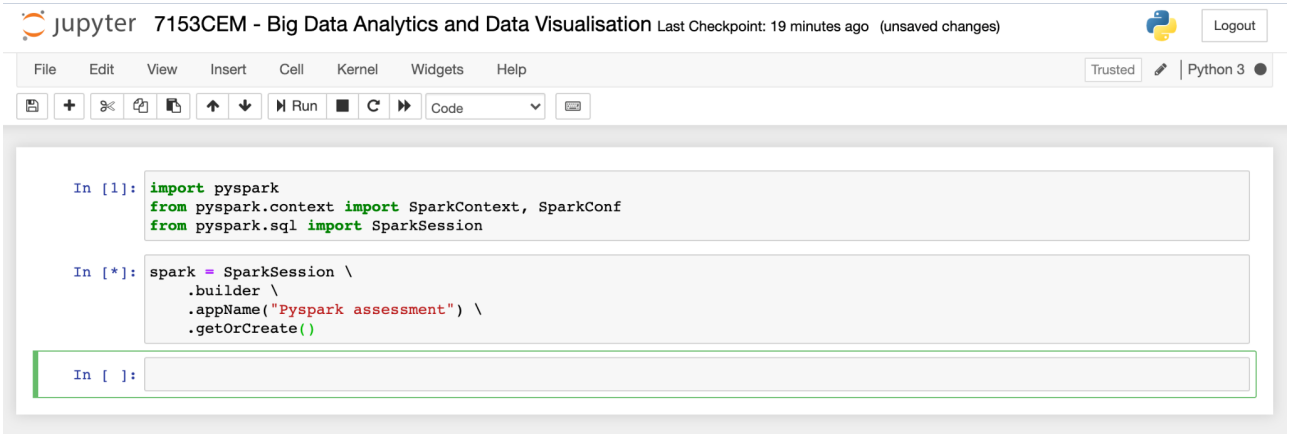
2.4.1 Data preprocessing

In order to develop an accurate model, the data has to be cleaned, normalized, and formatted prior to being fed into the model. The quality and speed of the preprocessing can have a significant impact on the model performance, often exceeding its architecture.

Spark is an open-source framework for fast data processing on large data sets across multiple computers. Spark is implemented in Scala and runs in JVM, but also has APIs for Java, Scala, Python, R, and SparkSQL.

Importing Pyspark & spark session

To start with, let's create import pyspark packages and a SparkSession.



Fig(2.4.1)

2.4.2 Loading the dataset

We can use various functions of the DataFrame to explore a dataset once it has been loaded into memory as a DataFrame. The show's selected dataset is saved in downloads and is loaded into a dataframe variable. 22 attributes from the dataset are shown in Fig (4.2.1).

```
dataframe = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/Life Expectancy Data.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
dataframe.show(2)
```

```
+-----------+----+----------+---------------+---------------+-------------+-------+----------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-------------------
-+------------------+----------------------------------+---------+
|    Country|Year|    Status|Life expectancy|Adult Mortality|infant deaths|Alcohol|percentage expenditure|Hepatitis B
|Measles| BMI|under-five deaths|Polio|Total expenditure|Diphtheria|HIV/AIDS|       GDP|Population|thinness  1-19 year
s|thinness 5-9 years|Income composition of resources|Schooling|
+-----------+----+----------+---------------+---------------+-------------+-------+----------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-------------------
-+------------------+----------------------------------+---------+
|Afghanistan|2015|Developing|             65|            263|           62|   0.01|           71.27962362|         65
|   1154|19.1|                83|    6|             8.16|         65|    0.1| 584.25921|  33736494|                17.
2|              17.3|                             0.479|     10.1|
|Afghanistan|2014|Developing|           59.9|            271|           64|   0.01|           73.52358168|         62
|    492|18.6|                86|   58|             8.18|         62|    0.1|612.696514|    327582|                17.
5|              17.5|                             0.476|       10|
+-----------+----+----------+---------------+---------------+-------------+-------+----------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-------------------
-+------------------+----------------------------------+---------+
only showing top 2 rows
```

Fig(4.2.1)

The chosen dataset does not contain continents of each country, so I preferred to add one more dataset for more detailed analysis. The countryContinent.csv contains 9 attributes are shown in Fig (4.2.2).

```
continent_df = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/countryContinent.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
continent_df.show(4)
```

```
+-------------+------+------+------------+-------------+---------+---------------+-----------+---------------+
|      country|code_2|code_3|country_code|   iso_3166_2|continent|     sub_region|region_code|sub_region_code|
+-------------+------+------+------------+-------------+---------+---------------+-----------+---------------+
| Afghanistan|    AF|   AFG|           4|ISO 3166-2:AF|     Asia|  Southern Asia|        142|             34|
|Åland Islands|    AX|   ALA|         248|ISO 3166-2:AX|   Europe|Northern Europe|        150|            154|
|      Albania|    AL|   ALB|           8|ISO 3166-2:AL|   Europe|Southern Europe|        150|             39|
|      Algeria|    DZ|   DZA|          12|ISO 3166-2:DZ|   Africa|Northern Africa|          2|             15|
+-------------+------+------+------------+-------------+---------+---------------+-----------+---------------+
only showing top 4 rows
```

Fig(4.2.2)

2.4.3 Pre-processing and Investigation

Merging Dataset

```
dataframe = dataframe.join(continent_df, dataframe.Country == continent_df.country ,"left") \
            .select(dataframe["*"],continent_df["continent"])
dataframe.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: string (nullable = true)
 |-- Status: string (nullable = true)
 |-- Life expectancy: string (nullable = true)
 |-- Adult Mortality: string (nullable = true)
 |-- infant deaths: string (nullable = true)
 |-- Alcohol: string (nullable = true)
 |-- percentage expenditure: string (nullable = true)
 |-- Hepatitis B: string (nullable = true)
 |-- Measles: string (nullable = true)
 |-- BMI: string (nullable = true)
 |-- under-five deaths: string (nullable = true)
 |-- Polio: string (nullable = true)
 |-- Total expenditure: string (nullable = true)
 |-- Diphtheria: string (nullable = true)
 |-- HIV/AIDS: string (nullable = true)
 |-- GDP: string (nullable = true)
 |-- Population: string (nullable = true)
 |-- thinness  1-19 years: string (nullable = true)
 |-- thinness 5-9 years: string (nullable = true)
 |-- Income composition of resources: string (nullable = true)
 |-- Schooling: string (nullable = true)
 |-- continent: string (nullable = true)
```

Fig(4.3.1)

It is very helpful to link countries to the correct continents using the join type, as illustrated above in figure Fig(4.3). The left join returns all rows from the left dataset regardless of whether or not a match is found on the right dataset, If no match is found, it assigns null to that record and drops records from the right.

Getting rid of unwanted space

Some attributes contain leading and trailing space. This needs to be changed because it could cause some errors in the future. For removing spaces, I used ltrim and rtrim.

```
# Remove leading and trailing space of the columnname
from pyspark.sql.functions import *
df = df.withColumn('BMI', ltrim(df.BMI))
df = df.withColumn('Measles', rtrim(df.Measles))
```

Fig(4.3.2)

After that, the values in the Status column are changed to True and False. False for "Developing" and True for "Developed".

```
import pyspark.sql.functions as f
dataframe = dataframe.withColumn(
        "Status",
        f.when(
            f.col("Status") == 'Developing',
            'False'
        ).when(
            f.col("Status") == 'Developed',
            'True'
        ).otherwise(f.col("Status").cast('string'))
    )
dataframe.show()
```

Fig(4.3.3)

A DataFrame can be explored by first understanding its schema.

```
: dataframe.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: string (nullable = true)
 |-- Status: string (nullable = true)
 |-- Life expectancy: string (nullable = true)
 |-- Adult Mortality: string (nullable = true)
 |-- infant deaths: string (nullable = true)
 |-- Alcohol: string (nullable = true)
 |-- percentage expenditure: string (nullable = true)
 |-- Hepatitis B: string (nullable = true)
 |-- Measles: string (nullable = true)
 |-- BMI: string (nullable = true)
 |-- under-five deaths: string (nullable = true)
 |-- Polio: string (nullable = true)
 |-- Total expenditure: string (nullable = true)
 |-- Diphtheria: string (nullable = true)
 |-- HIV/AIDS: string (nullable = true)
 |-- GDP: string (nullable = true)
 |-- Population: string (nullable = true)
 |-- thinness  1-19 years: string (nullable = true)
 |-- thinness 5-9 years: string (nullable = true)
 |-- Income composition of resources: string (nullable = true)
 |-- Schooling: string (nullable = true)
 |-- continent: string (nullable = true)
```

Fig(4.3.4)

As you can see, DataFrame contains every attribute as a string. We need to infer the schema. Import IntegertType, BooleanType and FloatType library to change the data types from string.

```
from pyspark.sql.types import IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.types import FloatType


dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe = dataframe.withColumn("Status",dataframe["Status"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19 years"].cast(FloatType()))
dataframe = dataframe.withColumn("thinness 5-9 years",dataframe["thinness 5-9 years"].cast(FloatType()))
dataframe = dataframe.withColumn("Income composition of resources",dataframe["Income composition of resources"].cast(Fl
dataframe = dataframe.withColumn("Schooling",dataframe["Schooling"].cast(FloatType()))
```

Fig(4.3.5)

By using printSchema, we can see the types of each attribute.

```
dataframe.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Status: boolean (nullable = true)
 |-- Life expectancy: float (nullable = true)
 |-- Adult Mortality: integer (nullable = true)
 |-- infant deaths: integer (nullable = true)
 |-- Alcohol: float (nullable = true)
 |-- percentage expenditure: float (nullable = true)
 |-- Hepatitis B: integer (nullable = true)
 |-- Measles: integer (nullable = true)
 |-- BMI: float (nullable = true)
 |-- under-five deaths: integer (nullable = true)
 |-- Polio: integer (nullable = true)
 |-- Total expenditure: float (nullable = true)
 |-- Diphtheria: integer (nullable = true)
 |-- HIV/AIDS: float (nullable = true)
 |-- GDP: float (nullable = true)
 |-- Population: integer (nullable = true)
 |-- thinness  1-19 years: float (nullable = true)
 |-- thinness 5-9 years: float (nullable = true)
 |-- Income composition of resources: float (nullable = true)
 |-- Schooling: float (nullable = true)
 |-- continent: string (nullable = true)
```

Fig(4.3.6)

Figure (4.3.8) shows the number of rows in the dataset.

```
num_rows = dataframe.count()
print("number of rows: ", num_rows)
```

```
number of rows:  2938
```

Fig(4.3.7)

To check every column a unique, distinct() function is used. Each column is unique here.

```
distinctDF = dataframe.distinct()
print("Distinct count: "+str(distinctDF.count()))
```

```
Distinct count: 2938
```

Fig(4.3.8)

Handling Null Values

Missing values must be handled carefully during the preprocessing of the dataset since many machine learning algorithms do not support them. An absence of values can be caused by data corruption or the failure to record data.

```
dataframe.filter(col("Country").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Year").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Status").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Life expectancy").isNull()).count()
```

```
10
```

```
dataframe.filter(col("Adult Mortality").isNull()).count()
```

```
10
```

```
dataframe.filter(col("infant deaths").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Alcohol").isNull()).count()
```

```
194
```

```
dataframe.filter(col("percentage expenditure").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Hepatitis B").isNull()).count()
```

```
553
```

Fig(4.3.9)

```
dataframe.filter(col("thinness  1-19 years").isNull()).count()
```

```
34
```

```
dataframe.filter(col("thinness 5-9 years").isNull()).count()
```

```
34
```

```
dataframe.filter(col("Income composition of resources").isNull()).count()
```

```
167
```

```
dataframe.filter(col("Schooling").isNull()).count()
```

```
163
```

```
dataframe.filter(col("Measles").isNull()).count()
```

```
0
```

```
dataframe.filter(col("BMI").isNull()).count()
```

```
34
```

```
dataframe.filter(col("under-five deaths").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Polio").isNull()).count()
```

```
19
```

```
dataframe.filter(col("Total expenditure").isNull()).count()
```

```
226
```

```
dataframe.filter(col("Diphtheria").isNull()).count()
```

```
19
```

```
dataframe.filter(col("HIV/AIDS").isNull()).count()
```

```
0
```

```
dataframe.filter(col("GDP").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Population").isNull()).count()
```

```
652
```

Fig(4.3.10)

Fig(4.3.9) and Fig(4.3.10) show the count of null values in each column.

fillna( ) can be used to replace missing values with 0.

```
dataframe=dataframe.fillna(value=0,subset=['Population'])
dataframe.filter(col("Population").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['GDP'])
dataframe.filter(col("GDP").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Income composition of resources'])
dataframe.filter(col("Income composition of resources").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Schooling'])
dataframe.filter(col("Schooling").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['thinness 5-9 years'])
dataframe.filter(col("thinness 5-9 years").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['thinness  1-19 years'])
dataframe.filter(col("thinness  1-19 years").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Diphtheria'])
dataframe.filter(col("Diphtheria").isNull()).count()
```

Fig(4.3.11)

```
dataframe=dataframe.fillna(value=0,subset=['Total expenditure'])
dataframe.filter(col("Total expenditure").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Polio'])
dataframe.filter(col("Polio").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Hepatitis B'])
dataframe.filter(col("Hepatitis B").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Alcohol'])
dataframe.filter(col("Alcohol").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Adult Mortality'])
dataframe.filter(col("Adult Mortality").isNull()).count()
```

```
0
```

```
dataframe=dataframe.fillna(value=0,subset=['Life expectancy'])
dataframe.filter(col("Life expectancy").isNull()).count()
```

```
0
```

Fig(4.3.12)

withColumnRenamed

Since we changed the Status column's data type from string to boolean. For better understanding, the column was renamed to "IsDeveloped".

```
dataframe = dataframe.withColumnRenamed("Status","IsDeveloped")
```

Fig(4.3.13)

```
dataframe.dtypes
```

```
[('Country', 'string'),
 ('Year', 'int'),
 ('IsDeveloped', 'boolean'),
 ('Life expectancy', 'float'),
 ('Adult Mortality', 'int'),
 ('infant deaths', 'int'),
 ('Alcohol', 'float'),
 ('percentage expenditure', 'float'),
 ('Hepatitis B', 'int'),
 ('Measles', 'int'),
 ('BMI', 'float'),
 ('under-five deaths', 'int'),
 ('Polio', 'int'),
 ('Total expenditure', 'float'),
 ('Diphtheria', 'int'),
 ('HIV/AIDS', 'float'),
 ('GDP', 'float'),
 ('Population', 'int'),
 ('thinness  1-19 years', 'float'),
 ('thinness 5-9 years', 'float'),
 ('Income composition of resources', 'float'),
 ('Schooling', 'float')]
```

Fig(4.3.14)

Fig(4.3.14) shows the dtype. The query retrieves PySpark DataFrame column names and data types(datatype).

Creating a CSV file

```
dataframe.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save("PreProcessedData.csv")
```

Fig(4.3.15)

2.5. Data Visualization

The visualization of data entails putting information into a visual context. Data visualization helps identify patterns and trends within large data sets. For plotting the dataset visually here I used Tableau software. We preprocessed the data and saved it as a CSV file before visualizing it. Then the csv file is opened on Tableau.

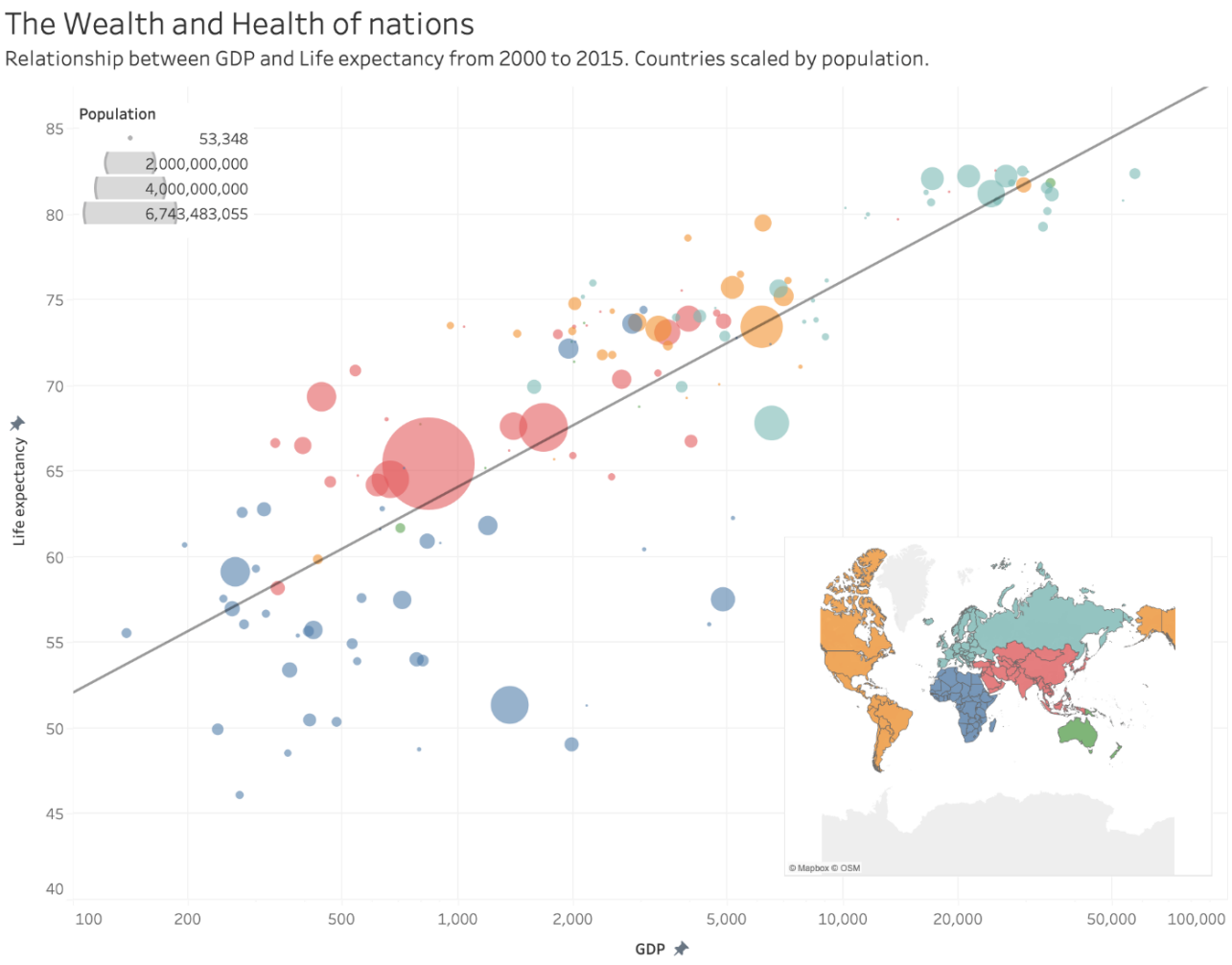


Fig (5.1)

Fig(5.1) show the relationship between the health and wealth of nations. The column represents average GDP and the row represents average Life expectancy. Each country is categorized on the basis of continents. The Color shows details about the continent. The countries are scaled by population. Size shows the sum of the population. The mark type is Circle. The data is filtered on Year, which keeps non-Null values only. The view is filtered on average Life expectancy, an average of GDP, Country and average Population. The average Life expectancy filter ranges from 10.40 to 82.54. The average GDP filter ranges from 1 to 57,363. The Country filter keeps 193 of 193 members. The sum of the Population filter ranges from 1 to 6,743,483,055.

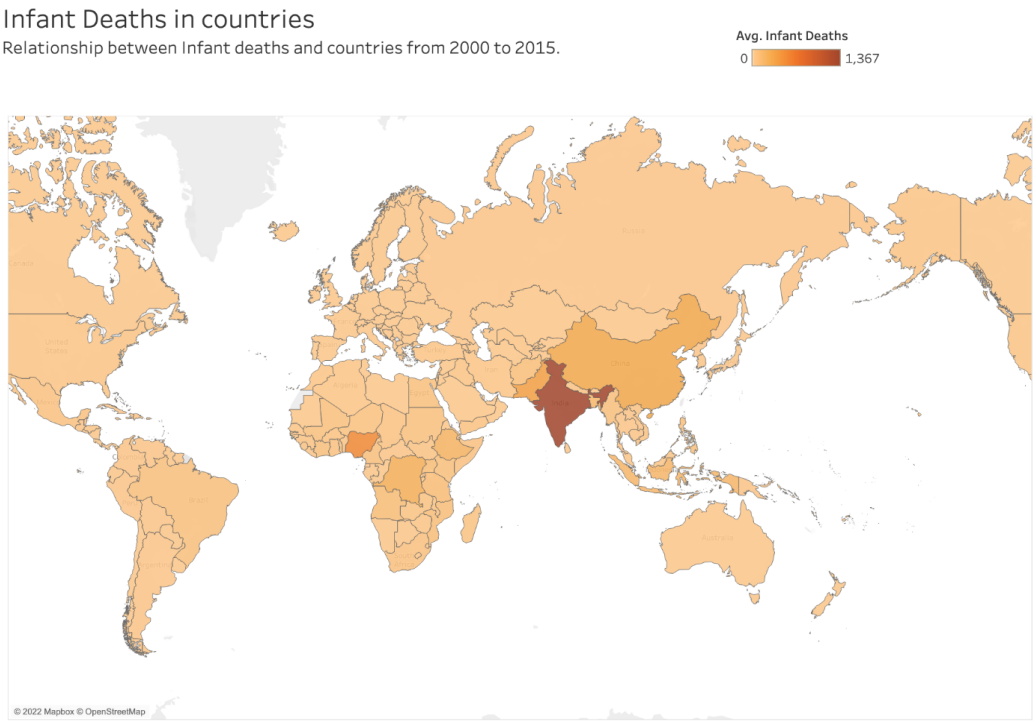


Fig (5.2)

Fig(5.2) is a map showing the relationship between infant deaths and countries from 2000 to 2015. Colour shows an average of Infant Deaths. Details are shown for Country. The mark type is Map.

The number of children under the age of one year that die is referred to as the infant mortality rate.

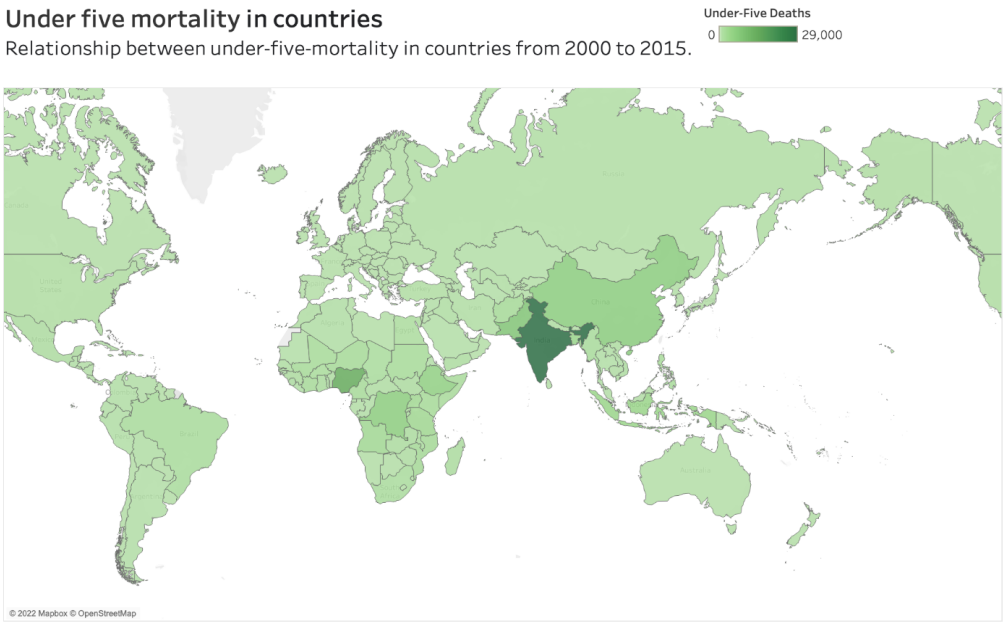


Fig (5.3)

Fig(5.3) shows the relationship between under-five deaths and countries from 2000 to 2015. Colour shows an average of Under-Five Deaths. Details are shown for Country. The mark type is Map.

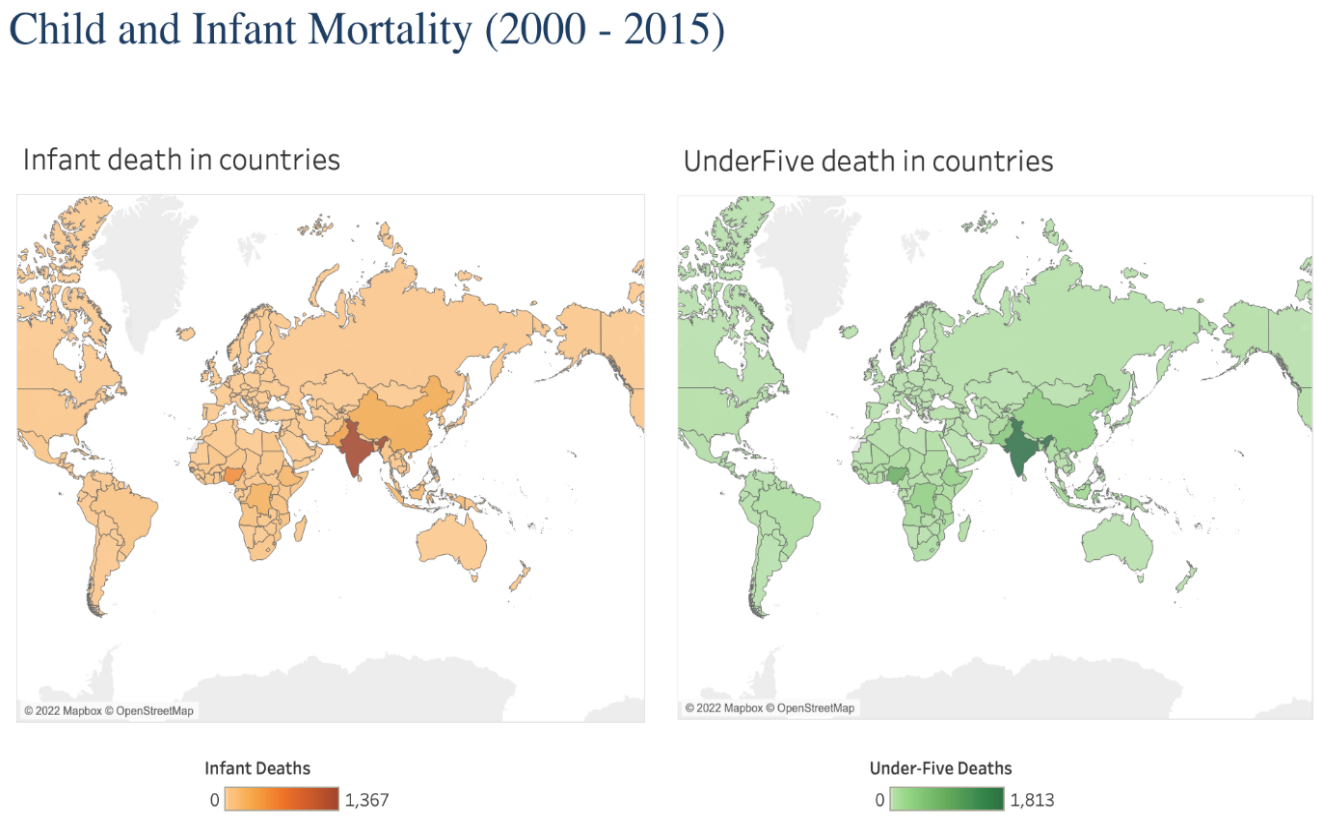


Fig (5.4)

Fig(5.4) shows the combined dashboard of infant mortality and under-five deaths. It is obvious from this map that India has a higher rate of child and infant mortality.

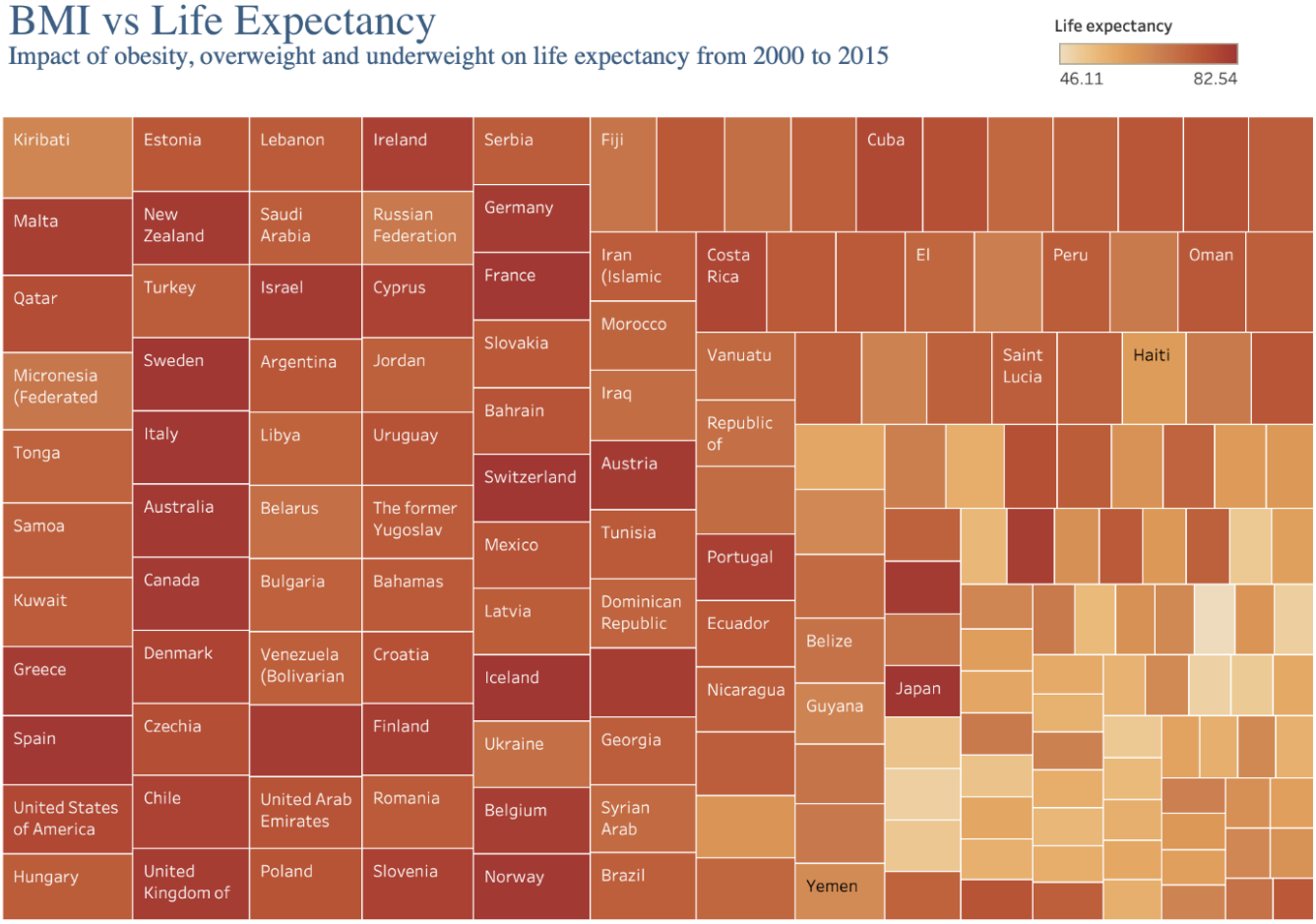


Fig (5.5) Treemap

Fig(5.5) shows the association between Body mass index and Life expectancy. The colour shows an average Life expectancy. Size shows an average BMI. The marks are labelled by Country. The view is filtered on average Life expectancy and average BMI. The average Life expectancy filter ranges from 1.00 to 82.54. The average BMI filter keeps non-Null values only.
From this Treemap it is understandable that BMI is affecting the Life expectancy of people. More people would live longer with less medical care if they control their BMI.

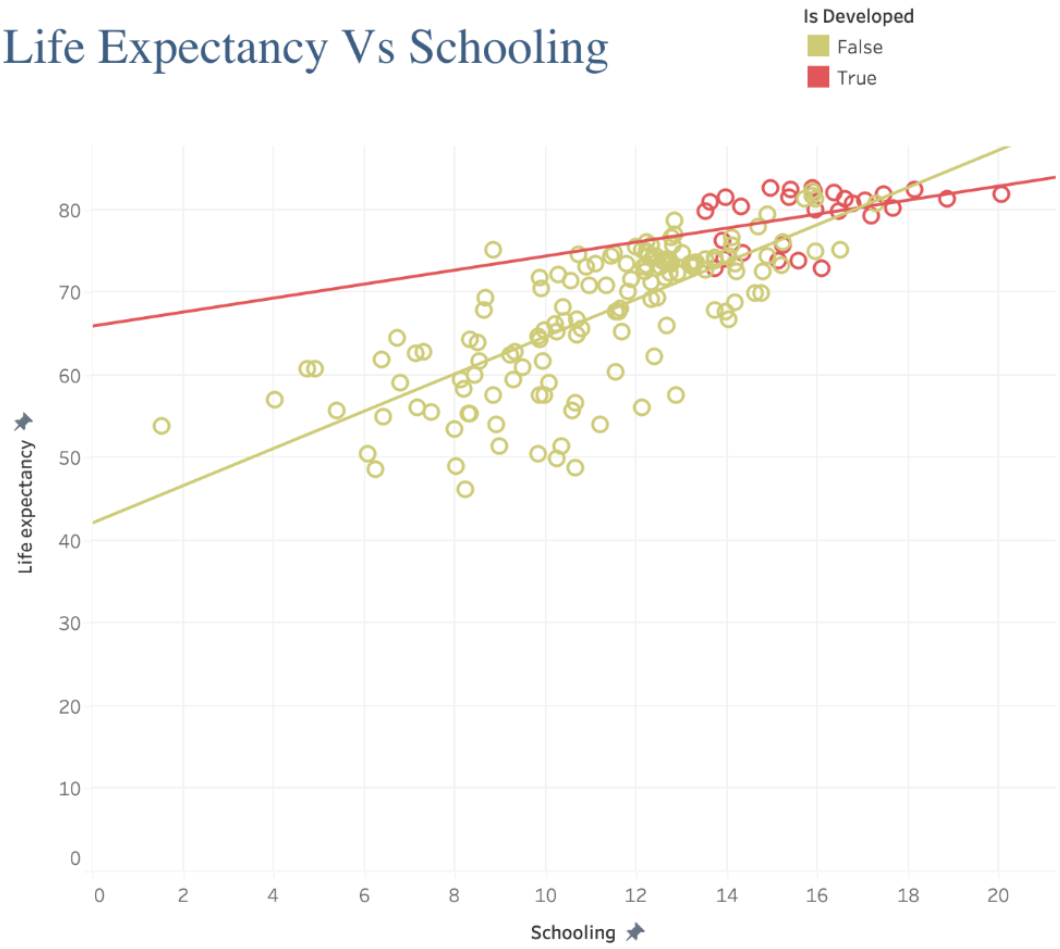


Fig (5.6)

Fig(5.6) describe the relationship between schooling and Life expectancy. Details are shown for countries. The colour shows the detail about whether the country is developed or not. The average Schooling filter ranges from 1.00 to 20.04. The average Life expectancy filter ranges from 1.10 to 82.54. A linear trend model is computed for average Life expectancy given the average Schooling. From this dashboard, we can comprehend that life expectancy is dependent on the education of the country.

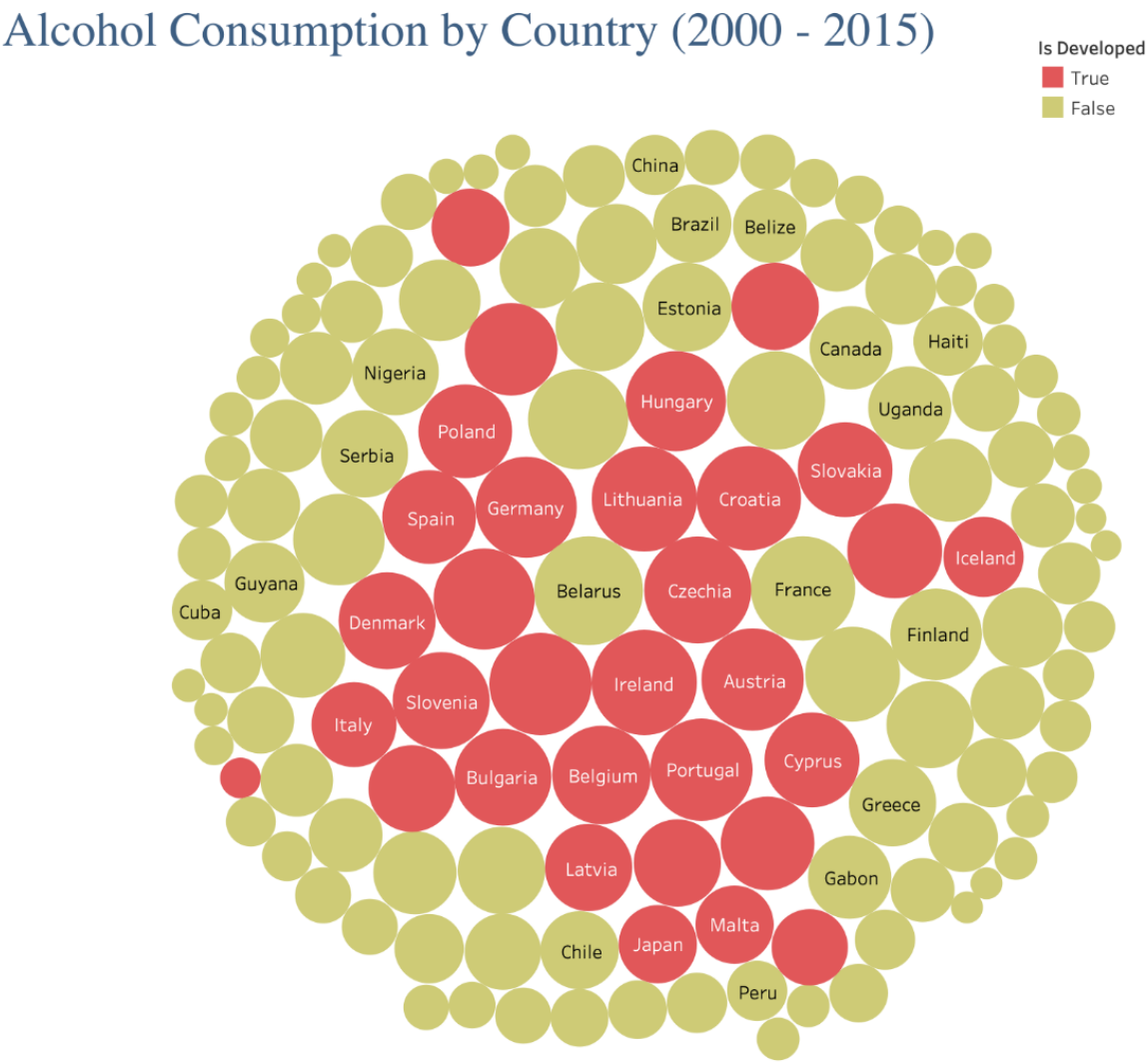


Fig (5.7) Packed Bubble

Fig (5.7) describe the average alcohol drinking habit in all developed and undeveloped nations. Colour shows details about whether the country is developed or not. Size indicates an average of Alcohol. The marks are labelled by Country. The view is filtered on average of Alcohol, which ranges from 1.00 to 12.65. Sorted descending by an average of Alcohol across all values of Country.

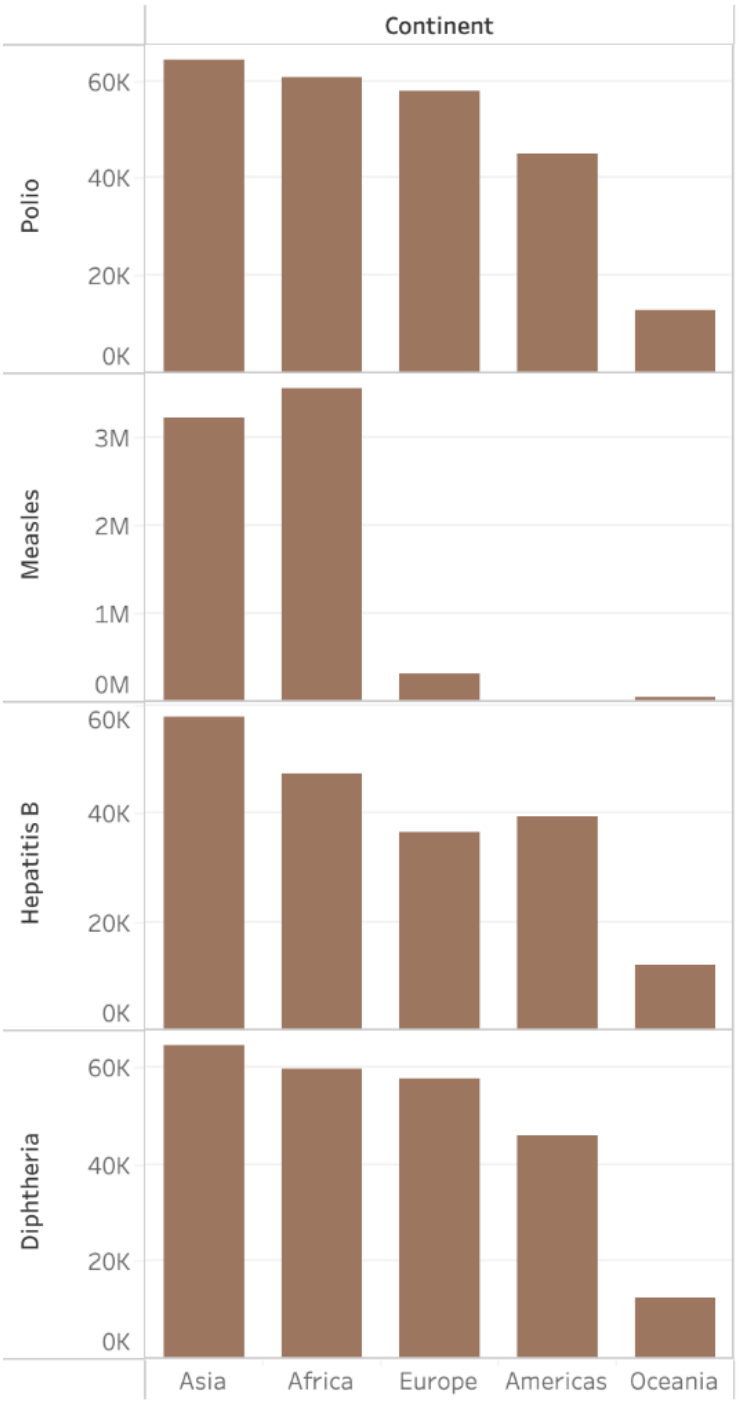


Fig (5.8) Bar chart

Fig(5.8) demonstrates the relation between continents and Hepatitis B, Measles, Diphtheria and Polio.

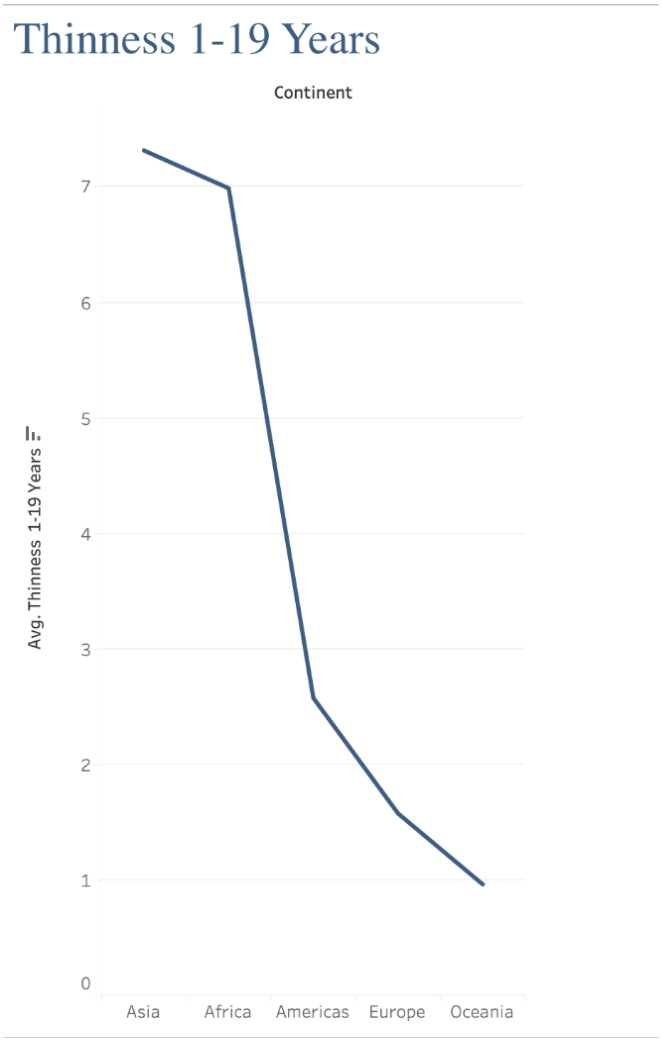


Fig (5.9)

The above figure Fig(5.9) shows the trend of Thinness 1-19 Years for Continent. The continent has 5 members on this sheet. Average of Thinness 1-19 Years range from 0.964 to 7.316 on this sheet.

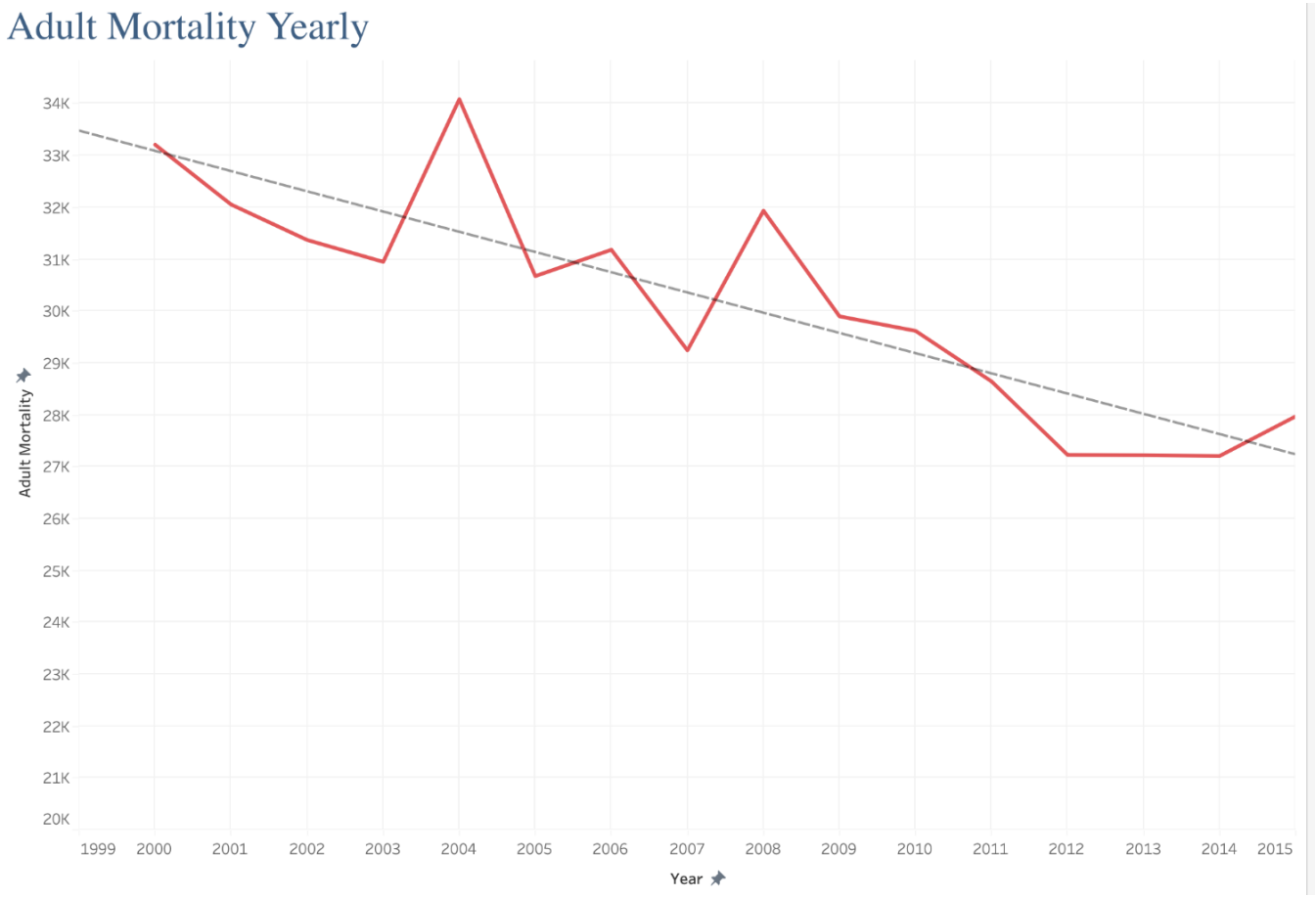


Fig (5.10)

The above figure illustrates the adult mortality rate from 2000 to 2015. There is a leap in adult mortality in the years 2004 and 2008. After that, we can see that adult mortality is gradually decreasing.

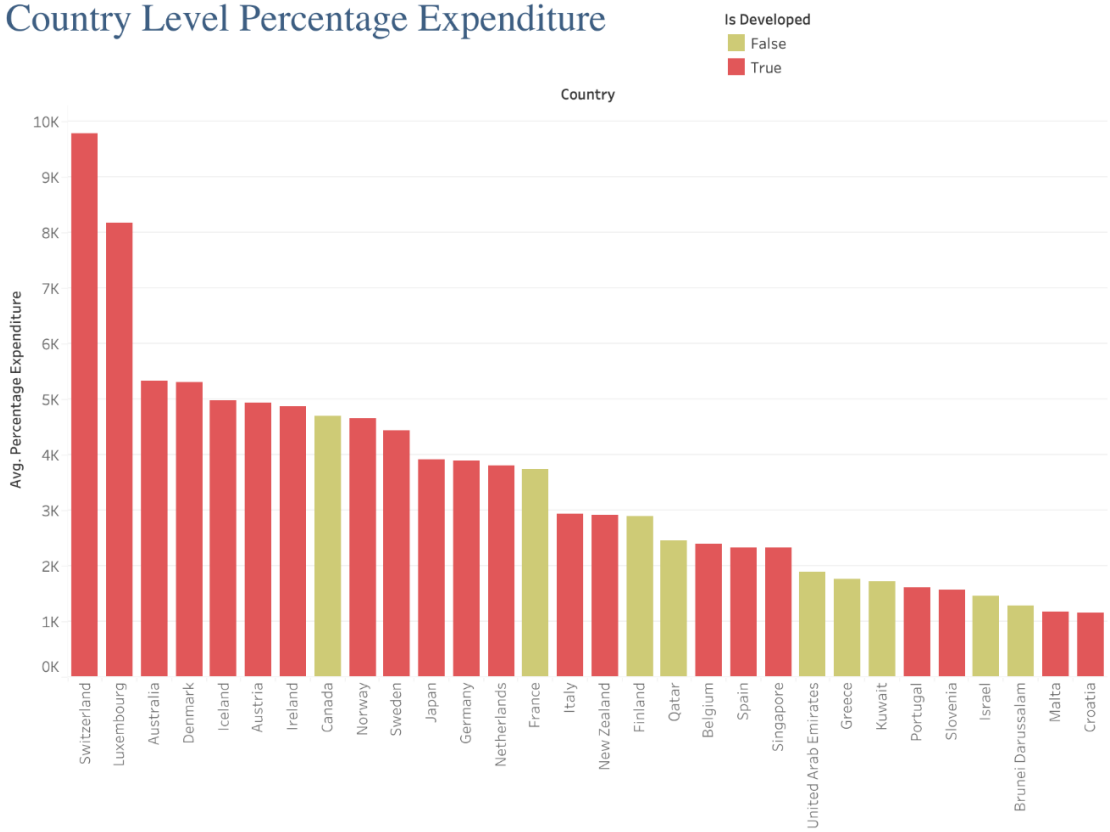


Fig (5.11)

Fig (5.11) shows the percentage expenditure of nations based on whether the country is developed or developing. The colour shows the status of nations. The graph shows only the top 30 countries with the most elevated percentage of expenditure.

2.6. Machine Learning

Machine learning is a branch of artificial intelligence (AI) (IBM Cloud Education, 2020). In Machine Learning, data and algorithms are used to mimic how humans learn, increasing their accuracy over time. It is a vital component of the rapidly expanding field of data science. In data mining projects, algorithms are trained to make predictions or classifications through the use of statistical methods.

Linear Regression

First I converted all string datatypes to appropriate datatype. The code used for converting string is given in Fig(6.1).

```
from pyspark.sql.types import IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.types import FloatType


dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe = dataframe.withColumn("IsDeveloped",dataframe["IsDeveloped"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19 years"].cast(FloatType()))
dataframe = dataframe.withColumn("thinness 5-9 years",dataframe["thinness 5-9 years"].cast(FloatType()))
dataframe = dataframe.withColumn("Income composition of resources",dataframe["Income composition of resources"].cast(Fl
dataframe = dataframe.withColumn("Schooling",dataframe["Schooling"].cast(FloatType()))
```

Fig (6.1)

Afterwards, I used printSchema to display the schema of dataframe.

```
df.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- IsDeveloped: boolean (nullable = true)
 |-- Life expectancy: float (nullable = true)
 |-- Adult Mortality: integer (nullable = true)
 |-- infant deaths: integer (nullable = true)
 |-- Alcohol: float (nullable = true)
 |-- percentage expenditure: float (nullable = true)
 |-- Hepatitis B: integer (nullable = true)
 |-- Measles: integer (nullable = true)
 |-- BMI: float (nullable = true)
 |-- under-five deaths: integer (nullable = true)
 |-- Polio: integer (nullable = true)
 |-- Total expenditure: float (nullable = true)
 |-- Diphtheria: integer (nullable = true)
 |-- HIV/AIDS: float (nullable = true)
 |-- GDP: float (nullable = true)
 |-- Population: integer (nullable = true)
 |-- thinness  1-19 years: float (nullable = true)
 |-- thinness 5-9 years: float (nullable = true)
 |-- Income composition of resources: float (nullable = true)
 |-- Schooling: float (nullable = true)
 |-- continent: string (nullable = true)
```

Fig (6.2)

Correlation

Correlation helps to identify the relationship between two attributes. Before identifying the correlation the dataframe and datatype are stored into a variable.

```
columns_data_type = df.dtypes
print(columns_data_type)
```

```
[('Country', 'string'), ('Year', 'int'), ('IsDeveloped', 'boolean'), ('Life expectancy', 'float'), ('Adult Mortalit
y', 'int'), ('infant deaths', 'int'), ('Alcohol', 'float'), ('percentage expenditure', 'float'), ('Hepatitis B', 'in
t'), ('Measles', 'int'), ('BMI', 'float'), ('under-five deaths', 'int'), ('Polio', 'int'), ('Total expenditure', 'flo
at'), ('Diphtheria', 'int'), ('HIV/AIDS', 'float'), ('GDP', 'float'), ('Population', 'int'), ('thinness  1-19 years',
'float'), ('thinness 5-9 years', 'float'), ('Income composition of resources', 'float'), ('Schooling', 'float'), ('co
ntinent', 'string')]
```

Fig (6.3)

The correlation between the values is identified by using the code below.

```
for column1_tuple in columns_data_type:
    if not column1_tuple[1] in ['float','int']:
        continue
    column_1_name = column1_tuple[0]
    for column2_tuple in columns_data_type:
        if not column2_tuple[1] in ['float','int']:
            continue
        column_2_name = column2_tuple[0]
        if column_1_name == column_2_name:
            continue

        coorelation = df.stat.corr(column_1_name,column_2_name)
        print("{}/{} = {}".format(column_1_name,column_2_name,coorelation))
```

```
Year/Life expectancy = 0.1364828767007698
Year/Adult Mortality = -0.08435627277187882
Year/infant deaths = -0.03717201444314333
Year/Alcohol = -0.1562655445686564
Year/percentage expenditure = 0.03172338000594529
Year/Hepatitis B = 0.3394632497381747
Year/Measles = -0.0828417004690695
Year/BMI = 0.10897364801746562
Year/under-five deaths = -0.04261799846774763
Year/Polio = 0.1056032651618374
Year/Total expenditure = -0.13507004941299797
Year/Diphtheria = 0.14497657769374722
Year/HIV/AIDS = -0.14064454168533566
Year/GDP = 0.0910377489622254
Year/Population = 0.013644169489855284
Year/thinness  1-19 years = -0.04787635997816759
Year/thinness 5-9 years = -0.05092905306816702
Year/Income composition of resources = 0.18773603696185193
Year/Schooling = 0.15456902694912378
```

Fig (6.4)

After that, I identified how many rows is obtainable for prediction with a positive correlation on selected attributes.

```
print( lin_df.count() )
lin_df = lin_df.filter(lin_df['Life expectancy']>0 ).filter(lin_df['GDP']>0 ).filter(lin_df['BMI']>0 )
lin_df = lin_df.filter(lin_df['Income composition of resources']>0 ).filter(lin_df['Schooling']>0 )
lin_df = lin_df.filter(lin_df['Polio']>0 ).filter(lin_df['Diphtheria']>0 )
print( lin_df.count() )
```

```
2904
2346
```

Fig (6.5)

Next using VectorAssembler I assigned an input and output value to an assembler. Following is the code for doing this.

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
#creating vectors from features
#Apache MLlib takes input if vector form
assembler=VectorAssembler(inputCols=['GDP','BMI','Polio','Diphtheria','Income composition of resources','Schooling'],ou
output=assembler.transform(lin_df)
output.select('features','Life expectancy').show(5)
#output as below
```

```
+--------------------+---------------+
|            features|Life expectancy|
+--------------------+---------------+
|[584.259216308593...|           65.0|
|[612.696533203125...|           59.9|
|[631.744995117187...|           59.9|
|[669.958984375,17...|           59.5|
|[63.5372314453125...|           59.2|
+--------------------+---------------+
only showing top 5 rows
```

Fig (6.6)

After that using a randomspilt, spilt the value for Life expectancy.

```
final_data=output.select('features','Life expectancy')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()
```

```
+-------+-----------------+
|summary|  Life expectancy|
+-------+-----------------+
|  count|             1674|
|   mean|69.91188771237609|
| stddev|9.745170751813951|
|    min|             36.3|
|    max|             89.0|
+-------+-----------------+
```

Fig (6.7)

```
test_data.describe().show()
```

```
+-------+-----------------+
|summary|  Life expectancy|
+-------+-----------------+
|  count|              672|
|   mean|69.15476186502548|
| stddev| 9.35518734481867|
|    min|             43.1|
|    max|             89.0|
+-------+-----------------+
```

Fig (6.8)

Now calculating prediction accuracy.

```
from pyspark.ml.regression import LinearRegression
linear_reg = LinearRegression(featuresCol='features',labelCol='Life expectancy')
trained_model = linear_reg.fit(train_data)
```

```
ship_results=trained_model.evaluate(train_data)
```

```
print('accuracy :',ship_results.r2)
```

```
accuracy : 0.8098253636472013
```

```
ship_results_test=trained_model.evaluate(test_data)
print('accuracy :',ship_results_test.r2)
```

```
accuracy : 0.7973664960535936
```

Fig (6.9)

Calculating the prediction.

```
unlabeled_data=test_data.select('features')
unlabeled_data.show(5)
```

```
+--------------------+
|            features|
+--------------------+
|[3.68594908714294...|
|[16.6399898529052...|
|[18.2532100677490...|
|[22.9944896697998...|
|[26.3932933807373...|
+--------------------+
only showing top 5 rows
```

Fig (6.9)

```
predictions=trained_model.transform(unlabeled_data)
predictions.show()
```

```
+--------------------+------------------+
|            features|        prediction|
+--------------------+------------------+
|[3.68594908714294...| 56.44149102097173|
|[11.3367795944213...| 58.62891556377875|
|[12.1789283752441...| 51.04267167140827|
|[12.2773303985595...| 77.09206806106428|
|[14.1422681808471...|  64.2419174536639|
|[17.7999496459960...|  50.4198294100943|
|[19.3177642822265...| 64.46634514023543|
|[21.5696544647216...| 52.00061629270833|
|[22.8548202514648...| 51.73280544288572|
|[24.2423229217529...| 53.55865505807201|
|[25.3837318420410...| 51.40628252733875|
|[26.3932933807373...| 56.37261315882886|
|[28.8321228027343...|56.256632734198945|
|[32.2227287292480...|53.542886594402034|
|[33.6674919128418...| 54.50556984148789|
|[37.1931762695312...| 53.43250690610837|
|[38.4657783508300...| 69.81329662060524|
|[39.4844741821289...|  61.4465446413158|
|[41.2699699401855...|  73.2564073457406|
|[41.8586311340332...| 56.65579366236743|
+--------------------+------------------+
only showing top 20 rows
```

Fig (6.10)

DecisionTreeRegressor

```
from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor(featuresCol='features',labelCol='Life expectancy')
model = dt.fit(train_data)
test_dt = model.transform(test_data)
test_dt.show(truncate=False)
```

```
+-----------------------------------------------------------------------------+---------------+------------------+
|features                                                                     |Life expectancy|prediction        |
+-----------------------------------------------------------------------------+---------------+------------------+
|[3.6859490871429443,12.100000381469727,0.4009999930858612,7.199999809265137] |57.7           |57.247887194996146|
|[11.336779594421387,2.799999952316284,0.4350000023841858,7.099999904632568]  |62.8           |57.247887194996146|
|[12.17892837524414,12.600000381469727,0.28299999237060547,4.300000190734863] |52.5           |53.11212146643436 |
|[12.27733039855957,59.900001525878906,0.796999990940094,14.699999809265137]  |74.8           |72.76000061035157 |
|[14.142268180847168,21.899999618530273,0.5210000276565552,7.5]               |65.1           |64.48714288984026 |
|[17.799949645996094,14.800000190734863,0.3089999854564667,7.199999809265137] |54.8           |53.11212146643436 |
|[19.317764282226562,15.399999618530273,0.5630000233650208,10.5]              |66.0           |64.48714288984026 |
|[21.56965446472168,13.699999809265137,0.3319999873638153,7.099999904632568]  |48.6           |53.11212146643436 |
|[22.854820251464844,14.399999618530273,0.30300000309944153,5.099999904632568]|48.1           |53.11212146643436 |
|[24.24232292175293,13.600000381469727,0.40400001406669617,11.0]              |48.8           |54.41454530195757 |
|[25.383731842041016,15.399999618530273,0.27799999713897705,3.5]              |53.7           |53.11212146643436 |
|[26.393293380737305,16.600000381469727,0.42500001192092896,9.300000190734863]|54.6           |54.41454530195757 |
|[28.832122802734375,17.5,0.42800000309944153,9.699999809265137]              |54.7           |54.41454530195757 |
|[32.22272872924805,16.600000381469727,0.3869999945163727,9.699999809265137]  |48.5           |54.41454530195757 |
|[33.6674919128418,24.0,0.4000000059604645,9.699999809265137]                 |59.2           |54.41454530195757 |
|[37.19317626953125,17.700000762939453,0.33799999952316284,5.900000095367432] |53.3           |53.11212146643436 |
|[38.46577835083008,45.099998474121094,0.6600000262260437,12.300000190734863] |71.4           |68.71890564344416 |
|[39.484474182128906,2.700000047683716,0.48399999737739563,7.599999904632568] |57.6           |61.81779673948126 |
|[41.26996994018555,5.099999904632568,0.7210000157356262,12.300000190734863]  |73.2           |73.05231778984827 |
|[41.8586311340332,15.199999809265137,0.44200000166893005,10.300000190734863] |55.5           |54.41454530195757 |
+-----------------------------------------------------------------------------+---------------+------------------+
only showing top 20 rows
```

Fig (6.10)

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life expectancy',metricName='r2')
print(evaluator.evaluate(test_dt))
```

```
0.8150208625170584
```

Fig (6.11)

```
train_dt = model.transform(train_data)
train_dt.show(truncate=False)
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life expectancy',metricName='r2')
print(evaluator.evaluate(train_dt))
```

```
+--------------------------------------------------------------------------+---------------+------------------+
|features                                                                  |Life expectancy|prediction        |
+--------------------------------------------------------------------------+---------------+------------------+
|[1.6813499927520752,18.700000762939453,0.625,11.399999618530273]          |66.8           |68.71890564344416 |
|[4.613574504852295,11.199999809265137,0.47600001096725464,7.699999809265137]|66.3           |61.81779673948126 |
|[5.668726444244385,17.100000381469727,0.43799999356269836,10.399999618530273]|68.0           |54.41454530195757 |
|[8.376432418823242,3.700000047683716,0.7400000095367432,14.0]             |73.3           |73.95098039215686 |
|[11.147276878356934,44.20000076293945,0.5770000219345093,10.699999809265137]|75.0           |68.71890564344416 |
|[11.55319595336914,19.200000762939453,0.6309999823570251,11.600000381469727]|67.2           |68.71890564344416 |
|[11.631377220153809,21.700000762939453,0.4490000009536743,7.699999809265137]|64.3           |57.247887194996146|
|[12.5664644241333,21.200000762939453,0.4440000057220459,7.5]              |63.5           |57.247887194996146|
|[12.989164352416992,23.799999237060547,0.4560000002384186,8.199999809265137]|53.3           |61.81779673948126 |
|[13.15419864654541,18.799999237060547,0.4790000021457672,10.899999618530273]|58.0           |58.42222213745117 |
|[14.214411735534668,39.0,0.6169999837875366,11.199999809265137]           |69.3           |68.71890564344416 |
|[15.574339866638184,14.100000381469727,0.28600001335144043,5.599999904632568]|53.4           |53.11212146643436 |
|[16.639989852905273,21.299999237060547,0.5139999985694885,7.300000190734863]|64.8           |64.48714288984026 |
|[17.81597137451172,31.600000381469727,0.5350000262260437,9.699999809265137]|64.0           |64.48714288984026 |
|[18.253210067749023,22.200000762939453,0.4550000131130218 5.7,9.900000095367432]|64.9           |61.81779673948126 |
|                                                                  .100000381469727]|59.9           |64.48714288984026 |
|[21.36238670349121,54.20000076293945,0.7200000286102295,12.5]             |65.1           |73.05231778984827 |
|[21.373329162597656,22.899999618530273,0.382999986410141,9.899999618530273]|57.9           |53.11212146643436 |
|[22.994489669799805,27.100000381469727,0.4930000071525574,9.100000381469727]|45.9           |61.81779673948126 |
|[24.819000244140625,55.70000076293945,0.828000009059906,14.899999618530273]|79.8           |77.31125011444092 |
+--------------------------------------------------------------------------+---------------+------------------+
only showing top 20 rows

0.845869669793683
```

Fig (6.12)

Decision Tree Classifier

Decision Tree Classifier are widely used since they are easy to interpret, can handle categorical features, support multi-class classification, do not require feature scaling.

Before starting we have to exclude data with null values. Using the code below I filtered the attributes of Life expectancy, GDP, BMI, Income composition of resources, Schooling, Total expenditure, percentage expenditure. The code is given below.

```
print( df.count() )
class_df = df
class_df = class_df.filter(class_df['Life expectancy']>0 ).filter(class_df['GDP']>0 ).filter(class_df['BMI']>0 )
class_df = class_df.filter(class_df['Income composition of resources']>0 ).filter(class_df['Schooling']>0 )
class_df = class_df.filter(class_df['Total expenditure']>0 ).filter(class_df['percentage expenditure']>0 )
print( class_df.count() )
```

```
2904
2195
```

Fig (6.13)

The above figure shows the total count of rows, which is 2904. After filtering it became 2195.

Then we combine all the feature columns into a single vector column using the VectorAssembler. The code is given below.

You can see that we now have a features column and a IsDeveloped column.

```
assembler=VectorAssembler(inputCols=['GDP','BMI','Income composition of resources', \
                                     'Schooling','Life expectancy','Total expenditure', \
                                     'percentage expenditure'],outputCol='features')
output=assembler.transform(class_df)
output.select('features','IsDeveloped').show(5)
```

```
+--------------------+-----------+
|            features|IsDeveloped|
+--------------------+-----------+
|[584.259216308593...|      false|
|[612.696533203125...|      false|
|[631.744995117187...|      false|
|[669.958984375,17...|      false|
|[63.5372314453125...|      false|
+--------------------+-----------+
only showing top 5 rows
```

Fig (6.14)

The decision trees are implemented to operate only numerical features but here the IsDeveloped column contains True or False values. We need to change to overcome future errors. The code is given below.

```
import pyspark.sql.functions as f
final_data=output.select('features','IsDeveloped')
final_data = final_data.withColumn(
        "IsDeveloped",
        f.when(
            f.col("IsDeveloped") == 'False', #Changing the False to 0
            0
        ).when(
            f.col("IsDeveloped") == 'True',  #Changing the True to 1
            1
        )
    )
final_data.count() #To check whether anydata is missing
```

```
2195
```

Fig (6.15)

Now randomly split data into train data and test data. The code is given below.

```
train_data_classifier,test_data__classifier=final_data.randomSplit([0.7,0.3])

train_data_classifier.describe().show()
test_data__classifier.describe().show()
```

```
+-------+-------------------+
|summary|        IsDeveloped|
+-------+-------------------+
|  count|               1502|
|   mean|0.19973368841544606|
| stddev|0.39993328347291196|
|    min|                  0|
|    max|                  1|
+-------+-------------------+

+-------+-------------------+
|summary|        IsDeveloped|
+-------+-------------------+
|  count|                693|
|   mean|0.17316017316017315|
| stddev|0.37865898765480527|
|    min|                  0|
|    max|                  1|
```

Fig (6.16)

```
train_data_classifier.count()
```

```
1538
```

```
test_data__classifier.count()
```

```
657
```

Fig (6.17)

Training Dataset Count: 1538

Test Dataset Count: 657

Importing DecisionTreeClassifier and fitting the train data to classifier model.

```
from pyspark.ml.classification import DecisionTreeClassifier

classifier = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'IsDeveloped', maxDepth=3)
classifier_Model = classifier.fit(train_data_classifier)
```

Fig (6.18)

Make prediction

```
predictions = classifier_Model.transform(test_data__classifier)
predictions.show()
```

```
+--------------------+-----------+-------------+--------------------+----------+
|            features|IsDeveloped|rawPrediction|         probability|prediction|
+--------------------+-----------+-------------+--------------------+----------+
|[1.68134999275207...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[4.61357450485229...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[11.3367795944213...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[12.2773303985595...|          1| [50.0,111.0]||[0.31055900621118...|       1.0|
|[14.1422681808471...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[14.2144117355346...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[15.5743398666381...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[18.2532100677490...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[19.6867179870605...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[24.8190002441406...|          1| [50.0,111.0]||[0.31055900621118...|       1.0|
|[24.9446525573730...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[25.3837318420410...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[26.1525173187255...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[26.3932933807373...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[27.5648250579834...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[32.2227287292480...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[33.4244995117187...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[33.6674919128418...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[39.2818336486816...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
|[42.7379646301269...|          0|[1099.0,43.0]||[0.96234676007005...|       0.0|
+--------------------+-----------+-------------+--------------------+----------+
only showing top 20 rows
```

Fig (6.18)

Evaluate our Decision Tree model.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="IsDeveloped", predictionCol="prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
Test Area Under ROC 0.924119630710089
```

Fig (6.19)

Test Area Under ROC is 0.924119630710089.

The decision tree performed well.

3. Discussion

The following discussion focuses on the visualization and machine learning results of this study. I made this analysis based on the data set on life expectancy that is available through the Global Health Observatory (GHO) data repository under the World Health Organization (WHO). The datasets are made open to the public for health data study. In my research, I have used three machine learning algorithms: linear regression, decision tree regression and decision tree classifier.

Firstly, I have found out the correlation between the attributes. The correlation measures the strength of the linear relationship between two quantitative variables. A positive correlation implies that as one variable increases, the other increases as well, and vice versa. It is evident from the results that the correlation between Life expectancy and GDP, BMI, Income composition of resources, and Schooling is a positive value, suggesting that these variables are positively related. I found 0.80 accuracies on training data, which is good. A positive linear relationship between life expectancy and GDP, BMI, Income composition of resources, and Schooling is shown. From this, we can understand that Life expectancy depends on the country's wealth, body mass index of people, Income and education. A better fund can be spent on medical care, research and education if the country has enough wealth.

With the same attributed I created another model and worked on decision tree regression. Here I got 0.81 accuracies. The same result is predicted on decision tree regression.

Secondly, I tried DecisionTreeClassifier. Here I used GDP, BMI, Income composition of resources, Schooling, Life expectancy, Total expenditure, percentage expenditure. I predicted these attributes using the IsDeveloped variable. I randomly split the data to test and train data. Using MulticlassClassificationEvaluator i got 0.92 accuracy. It is shown that developed countries have higher wealth, body mass index, Income, Education and Life expectancy.

4. Conclusion

As part of this project, data was pre-processed and visualised, then analysed, after that appropriate machine learning models applied. PySpark provides the best way for this project. All programs used in this report come with clear documentation. Overall, the results are satisfactory, but some uncertainties remain regarding the data quality and accuracy.

5. References

[1] Apache Spark. Spark.apache.org. Retrieved 06 February 2022, from https://spark.apache.org/

[2] Country to Continent. kaggle.com, Retrieved 06 February 2022, from https://www.kaggle.com/statchaitya/country-to-continent

[3] DecisionTreeClassifier. Spark.apache.org. Retrieved 10 February 2022, from https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.DecisionTreeClassifier.html

[4] Introduction to Machine Learning Algorithms: Linear Regression. towardsdatascience.com/. Retrieved 09 February 2022, from https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a

[5] Machine Learning. www.ibm.com. IBM Cloud Education 2020. Retrieved 12 February 2022, From https://www.ibm.com/uk-en/cloud/learn/machine-learning#:~:text=Machine%20learning%20is%20a%20branch,learn%2C%20gradually%20improving%20its%20accuracy.

[6] Life Expectancy (WHO) Kaggle.com. Retrieved 04 February 2022, from https://www.kaggle.com/kumarajarshi/life-expectancy-who

[7]Machine Learning Approaches and Its Applications. Medium.com. Retrieved 11 February 2022, from https://medium.datadriveninvestor.com/machine-learning-approaches-and-its-applications-7bfbe782f4a8

[8] Pyspark. Spark.apache.org. (2022). Retrieved 06 February 2022, from https://spark.apache.org/docs/latest/api/python/

[9] Tableau visualizations. towardsdatascience.com/. Retrieved 06 February 2022, from, https://towardsdatascience.com/tableau-visualizations-dc9e544dc9a8

6. Appendix

Appendix A - Installation

Java version

```
(base_env) Alans-MBP:Desktop aleenaalby$ java -version
java version "1.8.0_311"
Java(TM) SE Runtime Environment (build 1.8.0_311-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.311-b11, mixed mode)
```

Python version

```
Alans-MacBook-Pro:Desktop aleenaalby$ python -V
Python 3.9.5
```

Setting path

```
Alans-MacBook-Pro:Desktop aleenaalby$ export HADOOP_HOME=`pwd`/hadoop-2.7.3
Alans-MacBook-Pro:Desktop aleenaalby$ PATH=$HADOOP_HOME/bin:$PATH
Alans-MacBook-Pro:Desktop aleenaalby$ export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_311.jdk/Contents/Home
Alans-MacBook-Pro:Desktop aleenaalby$ PATH=$JAVA_HOME/bin:$PATH
```

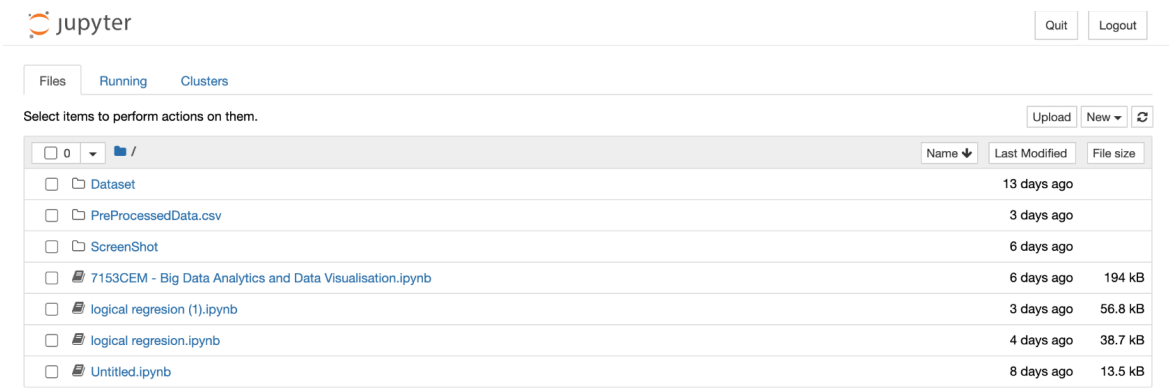
Pyspark installation

```
Alans-MacBook-Pro:~ aleenaalby$ pip3 install pyspark
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 32 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 26.1 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=281853646 sha256=ddca0e2f96a1a604e6c445eaca1e99687b7c26caf12562dd8c2ba62fea26ac4
2
  Stored in directory: /Users/aleenaalby/Library/Caches/pip/wheels/52/45/50/69db7b6e1da74a1b9fcc097827db9185cb8627117de852731e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

Setting jupyter notebook path

```
aleenaalby$ PYSPARK_DRIVER_PYTHON="jupyter"
aleenaalby$ export PYSPARK_DRIVER_PYTHON="jupyter"
aleenaalby$ export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
aleenaalby$  jupyter notebook
```

Jupyter Notebook



Appendix B - Data processing

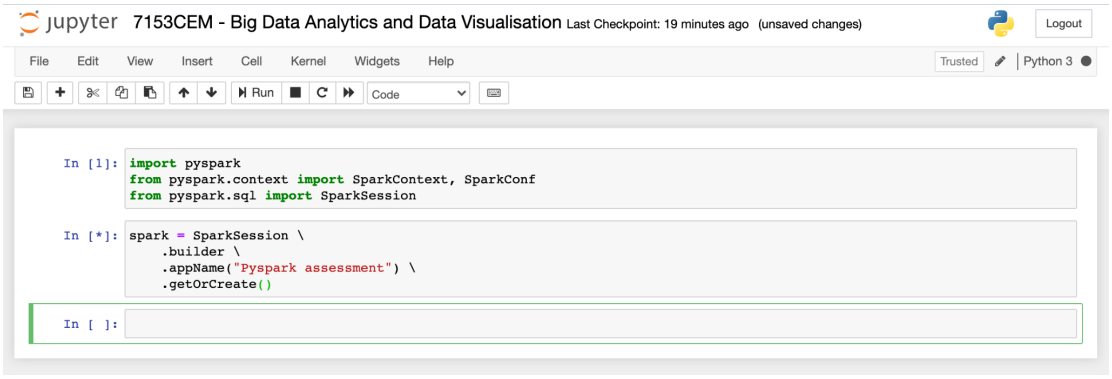
Datapreprocessing code

```
#Importing
import pyspark
from pyspark.context import SparkContext, SparkConf
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Pyspark assessment") \
    .getOrCreate()

#loading first dataset
dataframe = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/Life Expectancy
Data.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
dataframe.show(2)

#Loading second dataset
continent_df =
spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/countryContinent.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
continent_df.show(4)

#Joining two dataset
dataframe = dataframe.join(continent_df, dataframe.Country==continent_df.country ,"left") \
            .select(dataframe["*"],continent_df["continent"])
dataframe.printSchema()

#Remove leading and trailing space of the columnname
from pyspark.sql.functions import *
dataframe = dataframe.withColumn('BMI', ltrim(dataframe.BMI))
dataframe = dataframe.withColumn('Measles', rtrim(dataframe.Measles))

#Changing the status column values to True and False
import pyspark.sql.functions as f
dataframe = dataframe.withColumn(
        "Status",
        f.when(
            f.col("Status") == 'Developing',
```

```
            'False'
        ).when(
            f.col("Status") == 'Developed',
            'True'
        ).otherwise(f.col("Status").cast('string'))
    )
dataframe.show()

dataframe.printSchema()

#Changing the datatype
dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe = dataframe.withColumn("Status",dataframe["Status"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life
expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis
B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult
Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant
deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage
expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five
deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total
expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19
years"].cast(FloatType()))
dataframe = dataframe.withColumn("thinness 5-9 years",dataframe["thinness 5-9
years"].cast(FloatType()))
dataframe = dataframe.withColumn("Income composition of resources",dataframe["Income
composition of resources"].cast(FloatType()))
dataframe = dataframe.withColumn("Schooling",dataframe["Schooling"].cast(FloatType()))

#Counting number of rows
num_rows = dataframe.count()
print("number of rows: ", num_rows)
distinctDF = dataframe.distinct()
print("Distinct count: "+str(distinctDF.count()))
```

```
#Filling null values with value 0
dataframe=dataframe.fillna(value=0,subset=['Population'])
dataframe.filter(col("Population").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['GDP'])
dataframe.filter(col("GDP").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Income composition of resources'])
dataframe.filter(col("Income composition of resources").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Schooling'])
dataframe.filter(col("Schooling").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['thinness 5-9 years'])
dataframe.filter(col("thinness 5-9 years").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['thinness  1-19 years'])
dataframe.filter(col("thinness  1-19 years").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Diphtheria'])
dataframe.filter(col("Diphtheria").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Total expenditure'])
dataframe.filter(col("Total expenditure").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Polio'])
dataframe.filter(col("Polio").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Hepatitis B'])
dataframe.filter(col("Hepatitis B").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Alcohol'])
dataframe.filter(col("Alcohol").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Adult Mortality'])
dataframe.filter(col("Adult Mortality").isNull()).count()

dataframe=dataframe.fillna(value=0,subset=['Life expectancy'])
dataframe.filter(col("Life expectancy").isNull()).count()

#Renaming Column
dataframe = dataframe.withColumnRenamed("Status","IsDeveloped")
Dataframe.dtypes

#Writing to csv
dataframe.coalesce(1).write.format("com.databricks.spark.csv").option("header",
"true").save("PreProcessedData.csv")
```

Data preprocessing code and output




```
dataframe = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/Life Expectancy Data.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
dataframe.show(2)
```

```
+-----------+----+----------+---------------+---------------+-------------+-------+--------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-----------------
-+-----------------+--------------------------------+---------+
|     Country|Year|    Status|Life expectancy|Adult Mortality|infant deaths|Alcohol|percentage expenditure|Hepatitis B
|Measles| BMI|under-five deaths|Polio|Total expenditure|Diphtheria|HIV/AIDS|       GDP|Population|thinness  1-19 year
s|thinness 5-9 years|Income composition of resources|Schooling|
+-----------+----+----------+---------------+---------------+-------------+-------+--------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-----------------
-+-----------------+--------------------------------+---------+
|Afghanistan|2015|Developing|               65|            263|           62|   0.01|             71.27962362|         65
|   1154|19.1|                83|    6|             8.16|         65|    0.1| 584.25921|  33736494|                17.
2|               17.3|                           0.479|     10.1|
|Afghanistan|2014|Developing|               59.9|            271|           64|   0.01|             73.52358168|         62
|    492|18.6|                86|   58|             8.18|         62|    0.1|612.696514|    327582|                17.
5|               17.5|                           0.476|       10|
+-----------+----+----------+---------------+---------------+-------------+-------+--------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+----------+----------+-----------------
-+-----------------+--------------------------------+---------+
only showing top 2 rows
```

```
continent_df = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/Dataset/countryContinent.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
continent_df.show(4)
```

```
+-------------+------+------+------------+-------------+---------+--------------+-----------+--------------+
|      country|code_2|code_3|country_code|    iso_3166_2|continent|    sub_region|region_code|sub_region_code|
+-------------+------+------+------------+-------------+---------+--------------+-----------+--------------+
|  Afghanistan|    AF|   AFG|           4|ISO 3166-2:AF|     Asia|  Southern Asia|        142|            34|
|Åland Islands|    AX|   ALA|         248|ISO 3166-2:AX|   Europe|Northern Europe|        150|           154|
|      Albania|    AL|   ALB|           8|ISO 3166-2:AL|   Europe|Southern Europe|        150|            39|
|      Algeria|    DZ|   DZA|          12|ISO 3166-2:DZ|   Africa|Northern Africa|          2|            15|
+-------------+------+------+------------+-------------+---------+--------------+-----------+--------------+
only showing top 4 rows
```

```
dataframe.count()
```

```
2938
```

```
continent_df.count()
```

```
261
```

```
dataframe = dataframe.join(continent_df, dataframe.Country == continent_df.country ,"left") \
            .select(dataframe["*"],continent_df["continent"])
dataframe.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: string (nullable = true)
 |-- Status: string (nullable = true)
 |-- Life expectancy: string (nullable = true)
 |-- Adult Mortality: string (nullable = true)
 |-- infant deaths: string (nullable = true)
 |-- Alcohol: string (nullable = true)
 |-- percentage expenditure: string (nullable = true)
 |-- Hepatitis B: string (nullable = true)
 |-- Measles: string (nullable = true)
 |-- BMI: string (nullable = true)
 |-- under-five deaths: string (nullable = true)
 |-- Polio: string (nullable = true)
 |-- Total expenditure: string (nullable = true)
 |-- Diphtheria: string (nullable = true)
 |-- HIV/AIDS: string (nullable = true)
 |-- GDP: string (nullable = true)
 |-- Population: string (nullable = true)
 |-- thinness  1-19 years: string (nullable = true)
 |-- thinness 5-9 years: string (nullable = true)
 |-- Income composition of resources: string (nullable = true)
 |-- Schooling: string (nullable = true)
 |-- continent: string (nullable = true)
```

```
dataframe.show()
```

```
+-----------+----+----------+---------------+---------------+-------------+-------+----------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+-----------+----------+------------------
--+-----------------+-----------------------------+---------+---------+
|    Country|Year|    Status|Life expectancy|Adult Mortality|infant deaths|Alcohol|percentage expenditure|Hepatitis B
|Measles| BMI|under-five deaths|Polio|Total expenditure|Diphtheria|HIV/AIDS|        GDP|Population|thinness  1-19 yea
rs|thinness 5-9 years|Income composition of resources|Schooling|continent|
+-----------+----+----------+---------------+---------------+-------------+-------+----------------------+-----------
+-------+----+-----------------+-----+-----------------+----------+--------+-----------+----------+------------------
--+-----------------+-----------------------------+---------+---------+
|Afghanistan|2015|Developing|             65|            263|           62|   0.01|           71.27962362|         65
|   1154|19.1|               83|    6|             8.16|        65|     0.1|  584.25921|  33736494|                 1
7.2|             17.3|                        0.479|     10.1|     Asia|
|Afghanistan|2014|Developing|           59.9|            271|           64|   0.01|           73.52358168|         62
|    492|18.6|               86|   58|             8.18|        62|     0.1| 612.696514|    327582|                 1
7.5|             17.5|                        0.476|       10|     Asia|
|Afghanistan|2013|Developing|           59.9|            268|           66|   0.01|           73.21924272|         64
|    430|18.1|               89|   62|             8.13|        64|     0.1| 631.744976|  31731688|                 1
7.7|             17.7|                         0.47|      9.9|     Asia|
|Afghanistan|2012|Developing|           59.5|            272|           69|   0.01|            78.1842153|         67
```

```
# Remove leading and trailing space of the columnname
from pyspark.sql.functions import *
df = df.withColumn('BMI', ltrim(df.BMI))
df = df.withColumn('Measles', rtrim(df.Measles))
```

```
import pyspark.sql.functions as f
dataframe = dataframe.withColumn(
        "Status",
        f.when(
            f.col("Status") == 'Developing',
            'False'
        ).when(
            f.col("Status") == 'Developed',
            'True'
        ).otherwise(f.col("Status").cast('string'))
    )
dataframe.show()
```

```
from pyspark.sql.types import IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.types import FloatType


dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe = dataframe.withColumn("Status",dataframe["Status"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19 years"].cast(FloatType()))
dataframe = dataframe.withColumn("thinness 5-9 years",dataframe["thinness 5-9 years"].cast(FloatType()))
dataframe = dataframe.withColumn("Income composition of resources",dataframe["Income composition of resources"].cast(Fl
dataframe = dataframe.withColumn("Schooling",dataframe["Schooling"].cast(FloatType()))
```

```
num_rows = dataframe.count()
print("number of rows: ", num_rows)
```

```
number of rows:  2938
```

```
distinctDF = dataframe.distinct()
print("Distinct count: "+str(distinctDF.count()))
```

```
Distinct count: 2938
```

```
dataframe.filter(col("Country").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Year").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Status").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Life expectancy").isNull()).count()
```

```
10
```

```
dataframe.filter(col("Adult Mortality").isNull()).count()
```

```
10
```

```
dataframe.filter(col("infant deaths").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Alcohol").isNull()).count()
```

```
194
```

```
dataframe.filter(col("percentage expenditure").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Hepatitis B").isNull()).count()
```

```
553
```

```
dataframe.filter(col("thinness  1-19 years").isNull()).count()
```

```
34
```

```
dataframe.filter(col("thinness 5-9 years").isNull()).count()
```

```
34
```

```
dataframe.filter(col("Income composition of resources").isNull()).count()
```

```
167
```

```
dataframe.filter(col("Schooling").isNull()).count()
```

```
163
```

```
dataframe.filter(col("Measles").isNull()).count()
```

```
0
```

```
dataframe.filter(col("BMI").isNull()).count()
```

```
34
```

```
dataframe.filter(col("under-five deaths").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Polio").isNull()).count()
```

```
19
```

```
dataframe.filter(col("Total expenditure").isNull()).count()
```

```
226
```

```
dataframe.filter(col("Diphtheria").isNull()).count()
```

```
19
```

```
dataframe.filter(col("HIV/AIDS").isNull()).count()
```

```
0
```

```
dataframe.filter(col("GDP").isNull()).count()
```

```
0
```

```
dataframe.filter(col("Population").isNull()).count()
```

```
652
```

```
dataframe=dataframe.fillna(value=0,subset=['Population'])
dataframe.filter(col("Population").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['GDP'])
dataframe.filter(col("GDP").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Income composition of resources'])
dataframe.filter(col("Income composition of resources").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Schooling'])
dataframe.filter(col("Schooling").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['thinness 5-9 years'])
dataframe.filter(col("thinness 5-9 years").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['thinness  1-19 years'])
dataframe.filter(col("thinness  1-19 years").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Diphtheria'])
dataframe.filter(col("Diphtheria").isNull()).count()
```

```
dataframe=dataframe.fillna(value=0,subset=['Total expenditure'])
dataframe.filter(col("Total expenditure").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Polio'])
dataframe.filter(col("Polio").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Hepatitis B'])
dataframe.filter(col("Hepatitis B").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Alcohol'])
dataframe.filter(col("Alcohol").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Adult Mortality'])
dataframe.filter(col("Adult Mortality").isNull()).count()
```

0

```
dataframe=dataframe.fillna(value=0,subset=['Life expectancy'])
dataframe.filter(col("Life expectancy").isNull()).count()
```

0

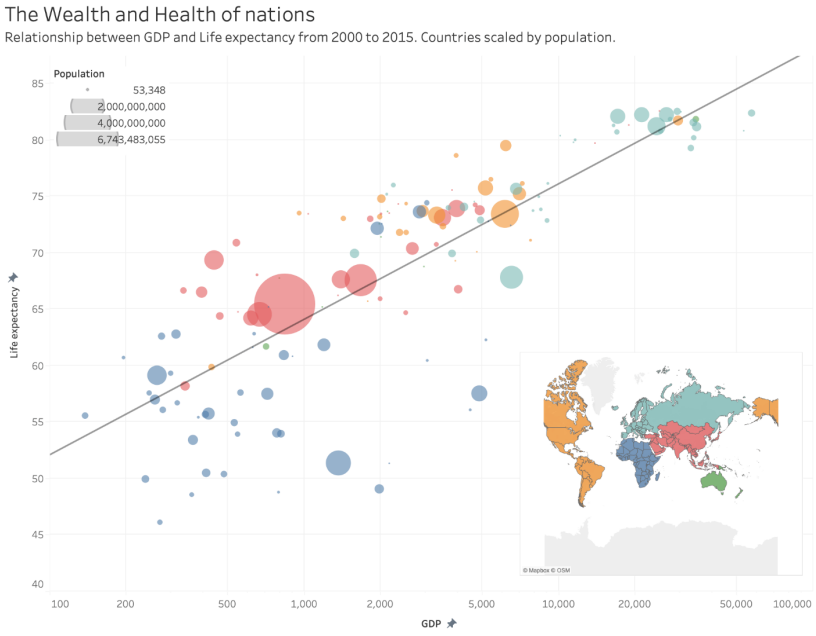
```
dataframe = dataframe.withColumnRenamed("Status","IsDeveloped")
```

```
dataframe.dtypes
```

```
[('Country', 'string'),
 ('Year', 'int'),
 ('IsDeveloped', 'boolean'),
 ('Life expectancy', 'float'),
 ('Adult Mortality', 'int'),
 ('infant deaths', 'int'),
 ('Alcohol', 'float'),
 ('percentage expenditure', 'float'),
 ('Hepatitis B', 'int'),
 ('Measles', 'int'),
 ('BMI', 'float'),
 ('under-five deaths', 'int'),
 ('Polio', 'int'),
 ('Total expenditure', 'float'),
 ('Diphtheria', 'int'),
 ('HIV/AIDS', 'float'),
 ('GDP', 'float'),
 ('Population', 'int'),
 ('thinness  1-19 years', 'float'),
 ('thinness 5-9 years', 'float'),
 ('Income composition of resources', 'float'),
 ('Schooling', 'float')]
```

```
dataframe.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save("PreProcessedData.csv")
```

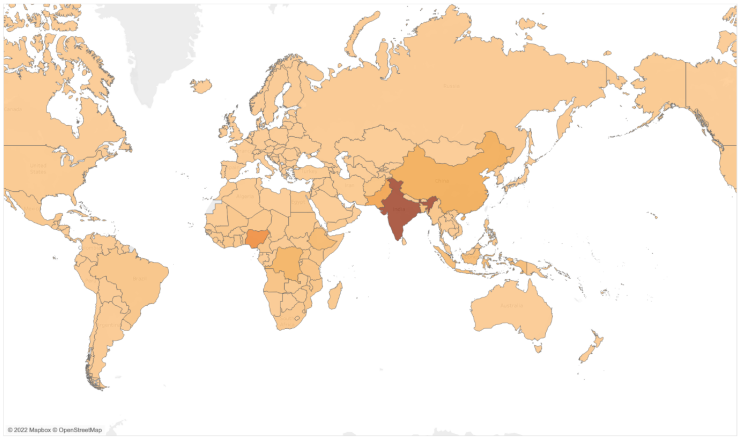
Appendix C - Data Visualization using Tableau



Infant Deaths in countries

Relationship between Infant deaths and countries from 2000 to 2015.

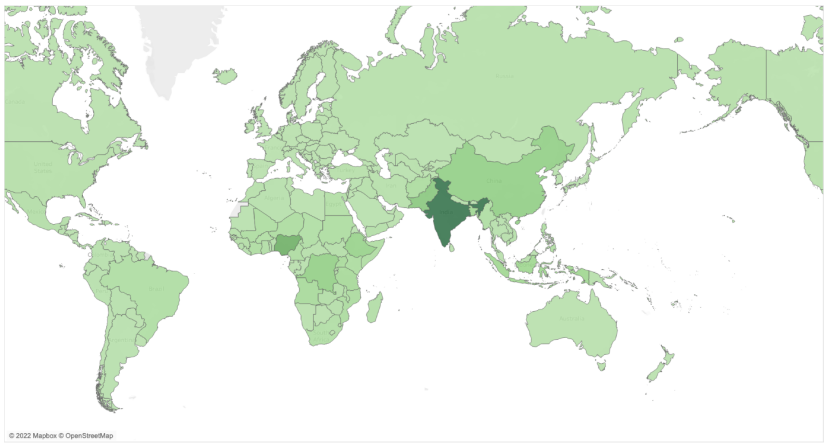
Avg. Infant Deaths
0 1,367



Under five mortality in countries

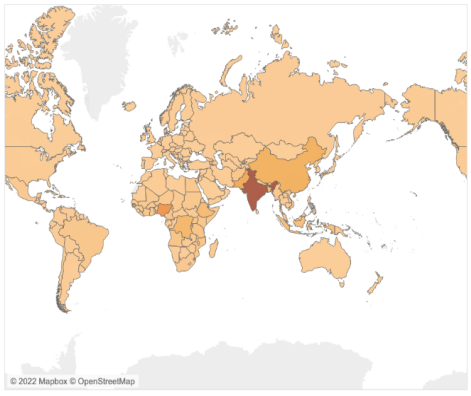
Relationship between under-five-mortality in countries from 2000 to 2015.

Under-Five Deaths
0 29,000



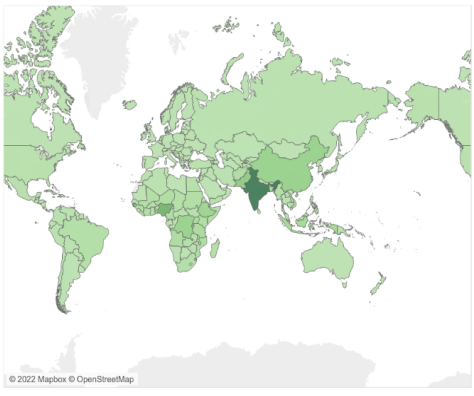
Child and Infant Mortality (2000 - 2015)

Infant death in countries



Infant Deaths
0 1,367

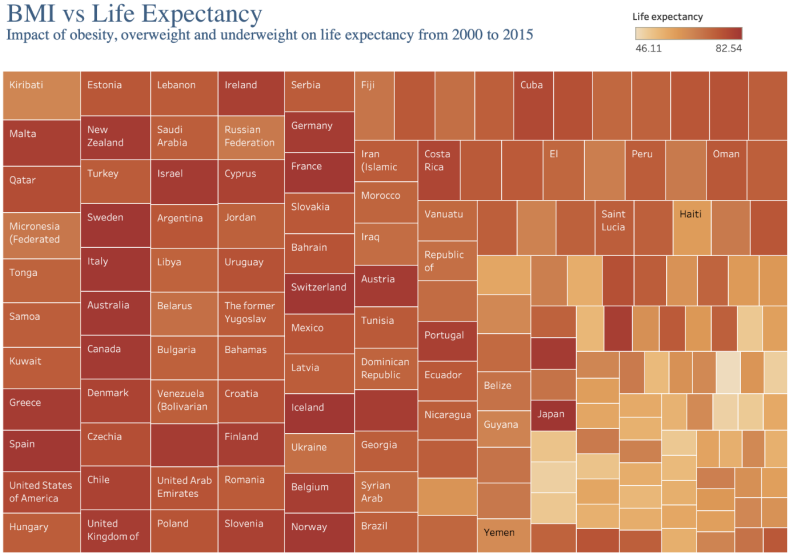
UnderFive death in countries



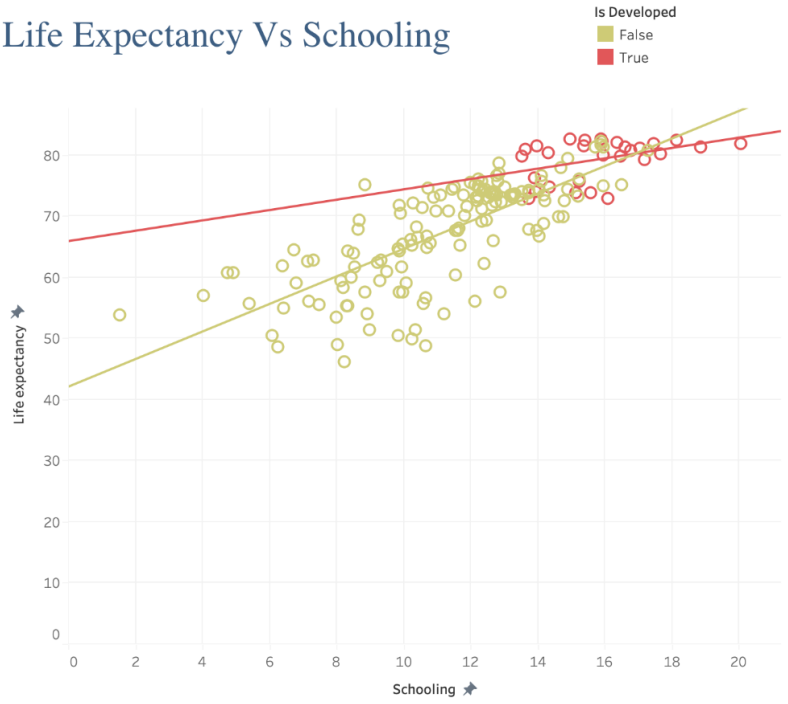
Under-Five Deaths
0 1,813

BMI vs Life Expectancy

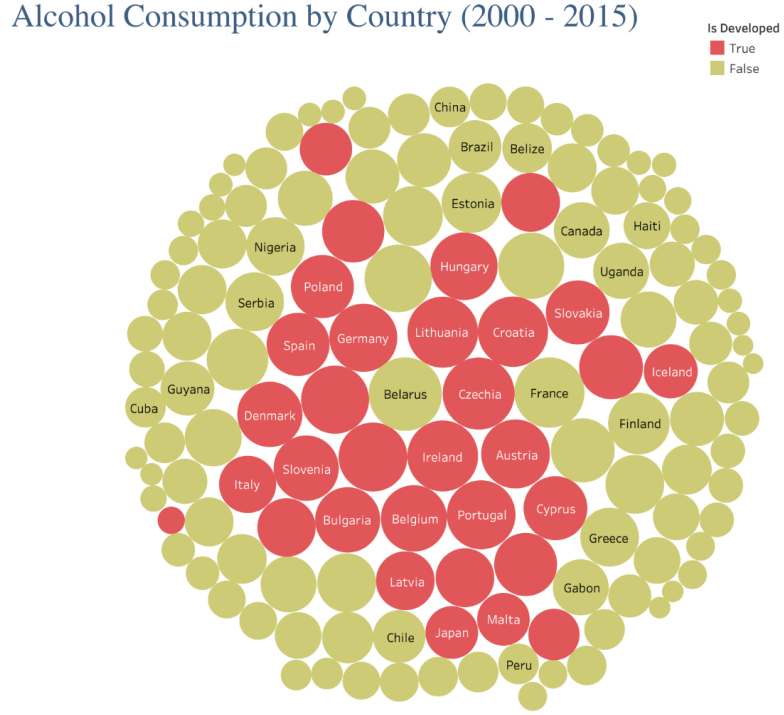
Impact of obesity, overweight and underweight on life expectancy from 2000 to 2015

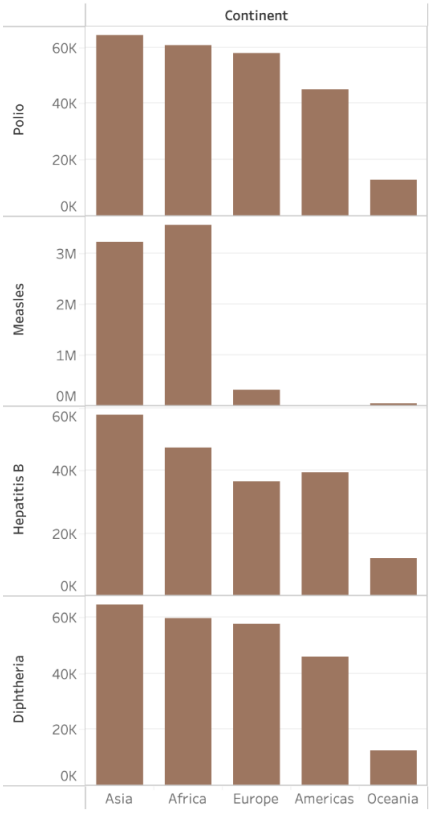


Life Expectancy Vs Schooling

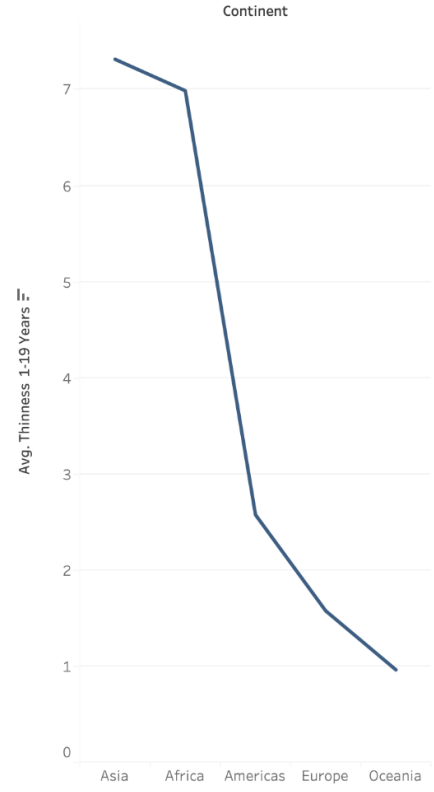


Alcohol Consumption by Country (2000 - 2015)

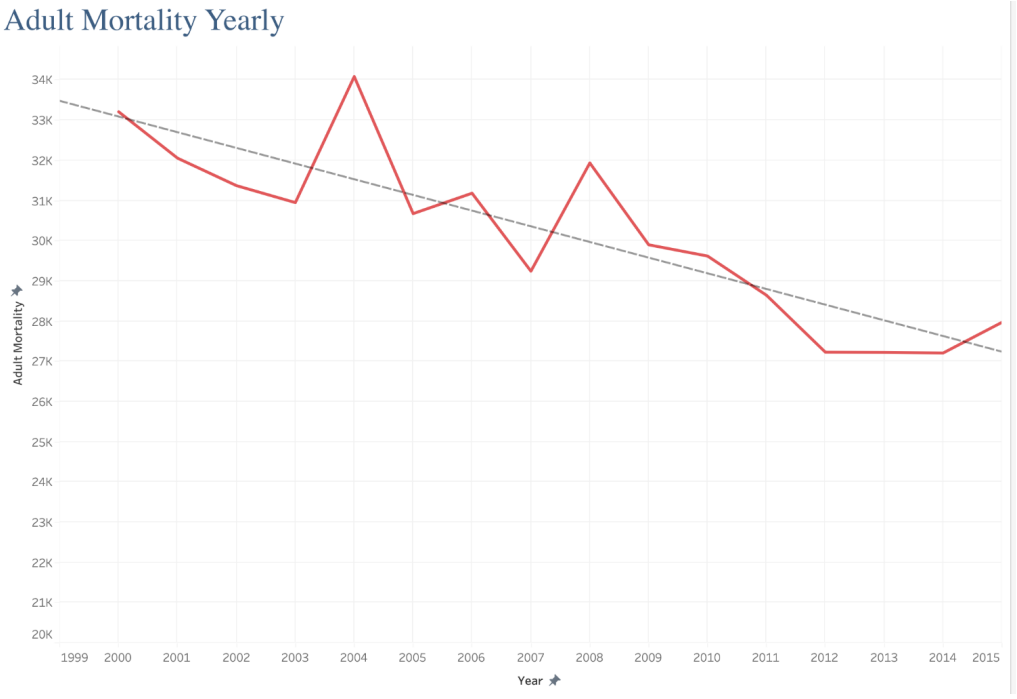


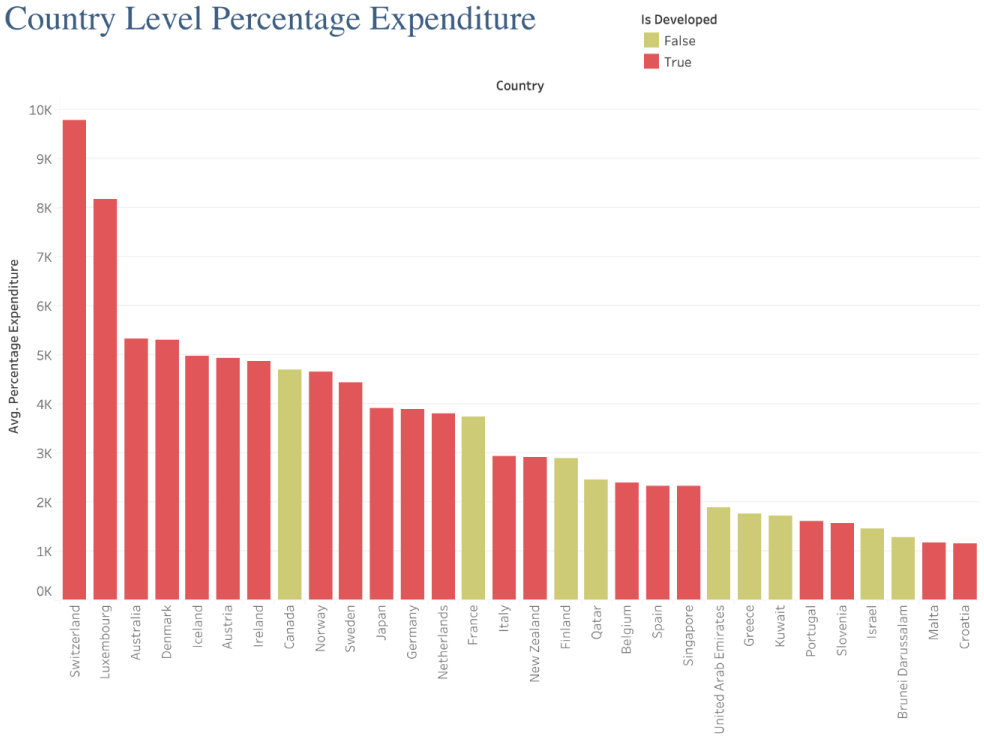


Thinness 1-19 Years



Adult Mortality Yearly





Appendix D - Linear Regression

```
#Import statement
import pyspark
from pyspark.context import SparkContext, SparkConf
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Pyspark assessment") \
    .getOrCreate()

#Loading preprocessed dataset
df =
spark.read.csv('/Users/aleenaalby/Documents/7153CEM/PreProcessedData.csv/part-00000-9058c94
e-6ba8-46c5-9563-c93dd3386755-c000.csv' \
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )

#Changing datatype
from pyspark.sql.types import IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.types import FloatType
```

```
dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe =
dataframe.withColumn("IsDeveloped",dataframe["IsDeveloped"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life
expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis
B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult
Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant
deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage
expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five
deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total
expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19
years"].cast(FloatType()))

df.printSchema()
df.columns
df = dataframe.dropna()

#Storing df type to columns_data_type variable
columns_data_type = df.dtypes
print(columns_data_type)

#computing correlation
for column1_tuple in columns_data_type:
    if not column1_tuple[1] in ['float','int']:
        continue
    column_1_name = column1_tuple[0]
    for column2_tuple in columns_data_type:
        if not column2_tuple[1] in ['float','int']:
            continue
        column_2_name = column2_tuple[0]
        if column_1_name == column_2_name:
```

```
            continue

        coorelation = df.stat.corr(column_1_name,column_2_name)
        print("{}/{} = {}".format(column_1_name,column_2_name,coorelation))


#filtering attribute with positive correlation
print( df.count() )
lin_df = df
lin_df = lin_df.filter(lin_df['Life expectancy']>0 ).filter(lin_df['GDP']>0
).filter(lin_df['BMI']>0 )
lin_df = lin_df.filter(lin_df['Income composition of resources']>0
).filter(lin_df['Schooling']>0 )
print( lin_df.count() )

#importing
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
#creating vectors from features
#Apache MLlib takes input if vector form
assembler=VectorAssembler(inputCols=['GDP','BMI','Income composition of
resources','Schooling'],outputCol='features')
output=assembler.transform(lin_df)
output.select('features','Life expectancy').show(5)
#output as below
final_data=output.select('features','Life expectancy')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()

from pyspark.ml.regression import LinearRegression
linear_reg = LinearRegression(featuresCol='features',labelCol='Life expectancy')
trained_model = linear_reg.fit(train_data)
ship_results=trained_model.evaluate(train_data)

print('accuracy :',ship_results.r2)
ship_results_test=trained_model.evaluate(test_data)
print('accuracy :',ship_results_test.r2)

unlabeled_data=test_data.select('features')
unlabeled_data.show(5)

predictions=trained_model.transform(unlabeled_data)
predictions.show()
test_data.show()
train_data.show()
```

```
import pyspark
from pyspark.context import SparkContext, SparkConf
from pyspark.sql import SparkSession
```

```
spark = SparkSession \
    .builder \
    .appName("Pyspark assessment") \
    .getOrCreate()
```

```
df = spark.read.csv('/Users/aleenaalby/Documents/7153CEM/PreProcessedData.csv/part-00000-9058c94e-6ba8-46c5-9563-c93dd3
                    , header=True, sep="," \
                    , multiLine=True \
                    , ignoreTrailingWhiteSpace=True \
                    , ignoreLeadingWhiteSpace=True )
```

```
from pyspark.sql.types import IntegerType
from pyspark.sql.types import BooleanType
from pyspark.sql.types import FloatType


dataframe = dataframe.withColumn("Year",dataframe["Year"].cast(IntegerType()))
dataframe = dataframe.withColumn("IsDeveloped",dataframe["IsDeveloped"].cast(BooleanType()))
dataframe = dataframe.withColumn("Life expectancy",dataframe["Life expectancy"].cast(FloatType()))
dataframe = dataframe.withColumn("Alcohol",dataframe["Alcohol"].cast(FloatType()))
dataframe = dataframe.withColumn("Hepatitis B",dataframe["Hepatitis B"].cast(IntegerType()))
dataframe = dataframe.withColumn("Polio",dataframe["Polio"].cast(IntegerType()))
dataframe = dataframe.withColumn("Adult Mortality",dataframe["Adult Mortality"].cast(IntegerType()))
dataframe = dataframe.withColumn("infant deaths",dataframe["infant deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Measles",dataframe["Measles"].cast(IntegerType()))
dataframe = dataframe.withColumn("percentage expenditure",dataframe["percentage expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("BMI",dataframe["BMI"].cast(FloatType()))
dataframe = dataframe.withColumn("under-five deaths",dataframe["under-five deaths"].cast(IntegerType()))
dataframe = dataframe.withColumn("Total expenditure",dataframe["Total expenditure"].cast(FloatType()))
dataframe = dataframe.withColumn("Diphtheria",dataframe["Diphtheria"].cast(IntegerType()))
dataframe = dataframe.withColumn("HIV/AIDS",dataframe["HIV/AIDS"].cast(FloatType()))
dataframe = dataframe.withColumn("GDP",dataframe["GDP"].cast(FloatType()))
dataframe = dataframe.withColumn("Population",dataframe["Population"].cast(IntegerType()))
dataframe = dataframe.withColumn("thinness  1-19 years",dataframe["thinness  1-19 years"].cast(FloatType()))
dataframe = dataframe.withColumn("thinness 5-9 years",dataframe["thinness 5-9 years"].cast(FloatType()))
dataframe = dataframe.withColumn("Income composition of resources",dataframe["Income composition of resources"].cast(Fl
dataframe = dataframe.withColumn("Schooling",dataframe["Schooling"].cast(FloatType()))
```

```
df.printSchema()
```

```
root
 |-- Country: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- IsDeveloped: boolean (nullable = true)
 |-- Life expectancy: float (nullable = true)
 |-- Adult Mortality: integer (nullable = true)
 |-- infant deaths: integer (nullable = true)
 |-- Alcohol: float (nullable = true)
 |-- percentage expenditure: float (nullable = true)
 |-- Hepatitis B: integer (nullable = true)
 |-- Measles: integer (nullable = true)
 |-- BMI: float (nullable = true)
 |-- under-five deaths: integer (nullable = true)
 |-- Polio: integer (nullable = true)
 |-- Total expenditure: float (nullable = true)
 |-- Diphtheria: integer (nullable = true)
 |-- HIV/AIDS: float (nullable = true)
 |-- GDP: float (nullable = true)
 |-- Population: integer (nullable = true)
 |-- thinness  1-19 years: float (nullable = true)
 |-- thinness 5-9 years: float (nullable = true)
 |-- Income composition of resources: float (nullable = true)
 |-- Schooling: float (nullable = true)
 |-- continent: string (nullable = true)
```

```
dataframe.show()
```

```
+-----------+----+----------+--------------+--------------+-------------+-------+---------------------+-----------
+-------+----+----------------+-----+----------------+----------+--------+-----------+----------+----------------
--+-----------------+------------------------------+---------+---------+
|    Country|Year|    Status|Life expectancy|Adult Mortality|infant deaths|Alcohol|percentage expenditure|Hepatitis B
|Measles| BMI|under-five deaths|Polio|Total expenditure|Diphtheria|HIV/AIDS|        GDP|Population|thinness  1-19 yea
rs|thinness 5-9 years|Income composition of resources|Schooling|continent|
+-----------+----+----------+--------------+--------------+-------------+-------+---------------------+-----------
+-------+----+----------------+-----+----------------+----------+--------+-----------+----------+----------------
--+-----------------+------------------------------+---------+---------+
|Afghanistan|2015|Developing|             65|           263|           62|   0.01|           71.27962362|         65
|   1154|19.1|                83|    6|            8.16|        65|    0.1|  584.25921|  33736494|                 1
7.2|             17.3|                         0.479|     10.1|     Asia|
|Afghanistan|2014|Developing|           59.9|           271|           64|   0.01|           73.52358168|         62
|    492|18.6|                86|   58|            8.18|        62|    0.1| 612.696514|    327582|                 1
7.5|             17.5|                         0.476|       10|     Asia|
|Afghanistan|2013|Developing|           59.9|           268|           66|   0.01|           73.21924272|         64
|    430|18.1|                89|   62|            8.13|        64|    0.1| 631.744976|  31731688|                 1
7.7|             17.7|                          0.47|      9.9|     Asia|
|Afghanistan|2012|Developing|           59.5|           272|           69|   0.01|            78.1842153|         67
```

```
columns_data_type = df.dtypes
print(columns_data_type)
```

```
[('Country', 'string'), ('Year', 'int'), ('IsDeveloped', 'boolean'), ('Life expectancy', 'float'), ('Adult Mortalit
y', 'int'), ('infant deaths', 'int'), ('Alcohol', 'float'), ('percentage expenditure', 'float'), ('Hepatitis B', 'in
t'), ('Measles', 'int'), ('BMI', 'float'), ('under-five deaths', 'int'), ('Polio', 'int'), ('Total expenditure', 'flo
at'), ('Diphtheria', 'int'), ('HIV/AIDS', 'float'), ('GDP', 'float'), ('Population', 'int'), ('thinness  1-19 years',
'float'), ('thinness 5-9 years', 'float'), ('Income composition of resources', 'float'), ('Schooling', 'float'), ('co
ntinent', 'string')]
```

```
for column1_tuple in columns_data_type:
    if not column1_tuple[1] in ['float','int']:
        continue
    column_1_name = column1_tuple[0]
    for column2_tuple in columns_data_type:
        if not column2_tuple[1] in ['float','int']:
            continue
        column_2_name = column2_tuple[0]
        if column_1_name == column_2_name:
            continue

        coorelation = df.stat.corr(column_1_name,column_2_name)
        print("{}/{} = {}".format(column_1_name,column_2_name,coorelation))
```

```
Year/Life expectancy = 0.1364828767007698
Year/Adult Mortality = -0.08435627277187882
Year/infant deaths = -0.03717201444314333
Year/Alcohol = -0.1562655445686564
Year/percentage expenditure = 0.03172338000594529
Year/Hepatitis B = 0.3394632497381747
Year/Measles = -0.0828417004690695
Year/BMI = 0.10897364801746562
Year/under-five deaths = -0.04261799846774763
Year/Polio = 0.1056032651618374
Year/Total expenditure = -0.13507004941299797
Year/Diphtheria = 0.14497657769374722
Year/HIV/AIDS = -0.14064454168533566
Year/GDP = 0.0910377489622254
Year/Population = 0.013644169489855284
Year/thinness  1-19 years = -0.04787635997816759
Year/thinness 5-9 years = -0.05092905306816702
Year/Income composition of resources = 0.18773603696185193
Year/Schooling = 0.15456902694912378
```

```
print( lin_df.count() )
lin_df = lin_df.filter(lin_df['Life expectancy']>0 ).filter(lin_df['GDP']>0 ).filter(lin_df['BMI']>0 )
lin_df = lin_df.filter(lin_df['Income composition of resources']>0 ).filter(lin_df['Schooling']>0 )
lin_df = lin_df.filter(lin_df['Polio']>0 ).filter(lin_df['Diphtheria']>0 )
print( lin_df.count() )
```

```
2904
2346
```

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
#creating vectors from features
#Apache MLlib takes input if vector form
assembler=VectorAssembler(inputCols=['GDP','BMI','Polio','Diphtheria','Income composition of resources','Schooling'],ou
output=assembler.transform(lin_df)
output.select('features','Life expectancy').show(5)
#output as below
```

```
+--------------------+---------------+
|            features|Life expectancy|
+--------------------+---------------+
|[584.259216308593...|           65.0|
|[612.696533203125...|           59.9|
|[631.744995117187...|           59.9|
|[669.958984375,17...|           59.5|
|[63.5372314453125...|           59.2|
+--------------------+---------------+
only showing top 5 rows
```

```
final_data=output.select('features','Life expectancy')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()
```

```
+-------+-----------------+
|summary|  Life expectancy|
+-------+-----------------+
|  count|             1674|
|   mean|69.91188771237609|
| stddev|9.745170751813951|
|    min|             36.3|
|    max|             89.0|
+-------+-----------------+
```

```
test_data.describe().show()
```

```
+-------+------------------+
|summary|  Life expectancy|
+-------+------------------+
|  count|               672|
|   mean|69.15476186502548|
| stddev| 9.35518734481867|
|    min|              43.1|
|    max|              89.0|
+-------+------------------+
```

```
from pyspark.ml.regression import LinearRegression
linear_reg = LinearRegression(featuresCol='features',labelCol='Life expectancy')
trained_model = linear_reg.fit(train_data)
```

```
ship_results=trained_model.evaluate(train_data)
```

```
print('accuracy :',ship_results.r2)
```

```
accuracy : 0.8098253636472013
```

```
ship_results_test=trained_model.evaluate(test_data)
print('accuracy :',ship_results_test.r2)
```

```
accuracy : 0.7973664960535936
```

```
unlabeled_data=test_data.select('features')
unlabeled_data.show(5)
```

```
+--------------------+
|            features|
+--------------------+
|[3.68594908714294...|
|[16.6399898529052...|
|[18.2532100677490...|
|[22.9944896697998...|
|[26.3932933807373...|
+--------------------+
only showing top 5 rows
```

```
predictions=trained_model.transform(unlabeled_data)
predictions.show()
```

```
+--------------------+------------------+
|            features|        prediction|
+--------------------+------------------+
|[3.68594908714294...| 56.44149102097173|
|[11.3367795944213...| 58.62891556377875|
|[12.1789283752441...| 51.04267167140827|
|[12.2773303985595...| 77.09206806106428|
|[14.1422681808471...|  64.2419174536639|
|[17.7999496459960...|  50.4198294100943|
|[19.3177642822265...| 64.46634514023543|
|[21.5696544647216...| 52.00061629270833|
|[22.8548202514648...| 51.73280544288572|
|[24.2423229217529...| 53.55865505807201|
|[25.3837318420410...| 51.40628252733875|
|[26.3932933807373...| 56.37261315882886|
|[28.8321228027343...|56.256632734198945|
|[32.2227287292480...|53.542886594402034|
|[33.6674919128418...| 54.50556984148789|
|[37.1931762695312...| 53.43250690610837|
|[38.4657783508300...| 69.81329662060524|
|[39.4844741821289...|  61.4465446413158|
|[41.2699699401855...|  73.2564073457406|
|[41.8586311340332...| 56.65579366236743|
+--------------------+------------------+
only showing top 20 rows
```

```
test_data.show()
```

```
+--------------------+---------------+
|            features|Life expectancy|
+--------------------+---------------+
|[3.68594908714294...|           57.7|
|[11.3367795944213...|           62.8|
|[12.1789283752441...|           52.5|
|[12.2773303985595...|           74.8|
|[14.1422681808471...|           65.1|
|[17.7999496459960...|           54.8|
|[19.3177642822265...|           66.0|
|[21.5696544647216...|           48.6|
|[22.8548202514648...|           48.1|
|[24.2423229217529...|           48.8|
|[25.3837318420410...|           53.7|
|[26.3932933807373...|           54.6|
|[28.8321228027343...|           54.7|
|[32.2227287292480...|           48.5|
|[33.6674919128418...|           59.2|
|[37.1931762695312...|           53.3|
|[38.4657783508300...|           71.4|
|[39.4844741821289...|           57.6|
|[41.2699699401855...|           73.2|
|[41.8586311340332...|           55.5|
+--------------------+---------------+
only showing top 20 rows
```

```
train_data.show()
```

```
+--------------------+---------------+
|            features|Life expectancy|
+--------------------+---------------+
|[1.68134999275207...|           66.8|
|[4.61357450485229...|           66.3|
|[5.66872644424438...|           68.0|
|[8.37643241882324...|           73.3|
|[11.1472768783569...|           75.0|
|[11.5531959533691...|           67.2|
|[11.6313772201538...|           64.3|
|[12.5664644241333...|           63.5|
|[12.9891643524169...|           53.3|
|[13.1541986465454...|           58.0|
|[14.2144117355346...|           69.3|
|[15.5743398666381...|           53.4|
|[16.6399898529052...|           64.8|
|[17.8159713745117...|           64.0|
|[18.2532100677490...|           64.9|
|[19.6867179870605...|           59.9|
|[21.3623867034912...|           65.1|
|[21.3733291625976...|           57.9|
|[22.9944896697998...|           45.9|
|[24.8190002441406...|           79.8|
+--------------------+---------------+
only showing top 20 rows
```

Appendix D - DecisionTreeRegressor

```
#Importing
from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor(featuresCol='features',labelCol='Life expectancy')
#Creating model
model = dt.fit(train_data)
```

```
test_dt = model.transform(test_data)
test_dt.show(truncate=False)

#Evaluating accurancy
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life
expectancy',metricName='r2')
print(evaluator.evaluate(test_dt))

train_dt = model.transform(train_data)
train_dt.show(truncate=False)
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life
expectancy',metricName='r2')
print(evaluator.evaluate(train_dt))
```

```
from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor(featuresCol='features',labelCol='Life expectancy')
model = dt.fit(train_data)
test_dt = model.transform(test_data)
test_dt.show(truncate=False)
```

```
+--------------------------------------------------------------------------------+---------------+------------------+
|features                                                                        |Life expectancy|prediction        |
+--------------------------------------------------------------------------------+---------------+------------------+
|[3.6859490871429443,12.100000381469727,0.4009999930858612,7.199999809265137]   |57.7           |57.247887194996146|
|[11.3367795944421387,2.799999952316284,0.4350000023841858,7.099999904632568]   |62.8           |57.247887194996146|
|[12.17892837524414,12.600000381469727,0.28299999237060547,4.300000190734863]   |52.5           |53.11212146643436 |
|[12.27733039855957,59.900001525878906,0.796999990940094,14.699999809265137]    |74.8           |72.76000061035157 |
|[14.142268180847168,21.899999618530273,0.5210000276565552,7.5]                 |65.1           |64.48714288984026 |
|[17.799949645996094,14.800000190734863,0.3089999854564667,7.199999809265137]   |54.8           |53.11212146643436 |
|[19.317764282226562,15.399999618530273,0.5630000233650208,10.5]                |66.0           |64.48714288984026 |
|[21.56965446472168,13.699999809265137,0.3319999873638153,7.099999904632568]    |48.6           |53.11212146643436 |
|[22.854820251464844,14.399999618530273,0.30300000309944153,5.099999904632568]  |48.1           |53.11212146643436 |
|[24.24232292175293,13.600000381469727,0.40400001406669617,11.0]                |48.8           |54.41454530195757 |
|[25.383731842041016,15.399999618530273,0.27799999713897705,3.5]                |53.7           |53.11212146643436 |
|[26.393293380737305,16.600000381469727,0.42500001192092896,9.300000190734863]  |54.6           |54.41454530195757 |
|[28.832122802734375,17.5,0.42800000309944153,9.699999809265137]                 |54.7           |54.41454530195757 |
|[32.22272872924805,16.600000381469727,0.3869999945163727,9.699999809265137]    |48.5           |54.41454530195757 |
|[33.6674919128418,24.0,0.4000000059604645,9.699999809265137]                    |59.2           |54.41454530195757 |
|[37.19317626953125,17.700000762939453,0.33799999952316284,5.900000095367432]   |53.3           |53.11212146643436 |
|[38.465778350830078,45.099998474121094,0.6600000262260437,12.300000190734863]  |71.4           |68.71890564344416 |
|[39.484474182128906,2.700000047683716,0.48399999737739563,7.599999904632568]   |57.6           |61.81779673948126 |
|[41.26996994018555,5.099999904632568,0.7210000157356262,12.300000190734863]    |73.2           |73.05231778984827 |
|[41.8586311340332,15.199999809265137,0.44200000166893005,10.300000190734863]   |55.5           |54.41454530195757 |
+--------------------------------------------------------------------------------+---------------+------------------+
only showing top 20 rows
```

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life expectancy',metricName='r2')
print(evaluator.evaluate(test_dt))
```

```
0.8150208625170584
```

```
train_dt = model.transform(train_data)
train_dt.show(truncate=False)
evaluator = RegressionEvaluator(predictionCol='prediction',labelCol='Life expectancy',metricName='r2')
print(evaluator.evaluate(train_dt))
```

```
+--------------------------------------------------------------------------------+---------------+------------------+
|features                                                                        |Life expectancy|prediction        |
+--------------------------------------------------------------------------------+---------------+------------------+
|[1.6813499927520752,18.700000762939453,0.625,11.399999618530273]                |66.8           |68.71890564344416 |
|[4.613574504852295,11.199999809265137,0.47600001096725464,7.699999809265137]   |66.3           |61.81779673948126 |
|[5.668726444244385,17.100000381469727,0.43799999356269836,10.399999618530273]  |68.0           |54.41454530195757 |
|[8.376432418823242,3.700000047683716,0.7400000095367432,14.0]                   |73.3           |73.95098039215686 |
|[11.147276878356934,44.20000076293945,0.5770000219345093,10.699999809265137]   |75.0           |68.71890564344416 |
|[11.55319595336914,19.200000762939453,0.6309999823570251,11.600000381469727]   |67.2           |68.71890564344416 |
|[11.631377220153809,21.700000762939453,0.4490000009536743,7.699999809265137]   |64.3           |57.247887194996146|
|[12.5664644241333,21.200000762939453,0.4440000057220459,7.5]                    |63.5           |57.247887194996146|
|[12.989164352416992,23.799999237060547,0.4560000002384186,8.199999809265137]   |53.3           |61.81779673948126 |
|[13.15419864654541,18.799999237060547,0.4790000021457672,10.899999618530273]   |58.0           |58.42222213745117 |
|[14.214411735534668,39.0,0.6169999837875366,11.199999809265137]                 |69.3           |68.71890564344416 |
|[15.574339866638184,14.100000381469727,0.28600001335144043,5.599999904632568]  |53.4           |53.11212146643436 |
|[16.639989852905273,21.299999237060547,0.5139999985694885,7.300000190734863]   |64.8           |64.48714288984026 |
|[17.81597137451172,31.600000381469727,0.5350000262260437,9.699999809265137]    |64.0           |64.48714288984026 |
|[18.253210067749023,22.200000762939453,0.45500001311302185,7.900000095367432]  |64.9           |61.81779673948126 |
|[19.686717987060547,23.600000381469727,0.5189999938011169,9.100000381469727]   |59.9           |64.48714288984026 |
|[21.362386703491211,54.20000076293945,0.7200000286102295,12.5]                  |65.1           |73.05231778984827 |
|[21.373329162597656,22.899999618530273,0.3829999864101410,9.899999618530273]   |57.9           |53.11212146643436 |
|[22.994489669799805,27.100000381469727,0.49300000071525574,9.100000381469727]  |45.9           |61.81779673948126 |
|[24.819000244140625,55.70000076293945,0.8280000090599060,14.899999618530273]   |79.8           |77.31125011444092 |
+--------------------------------------------------------------------------------+---------------+------------------+
only showing top 20 rows

0.845869669793683
```

Appendix E -Decision Tree Classifier

```
#Counting number of rows with value greater than 0 in selected attributes
print( df.count() )
class_df = df
class_df = class_df.filter(class_df['Life expectancy']>0 ).filter(class_df['GDP']>0
).filter(class_df['BMI']>0 )
class_df = class_df.filter(class_df['Income composition of resources']>0
).filter(class_df['Schooling']>0 )
class_df = class_df.filter(class_df['Total expenditure']>0 ).filter(class_df['percentage
expenditure']>0 )
print( class_df.count() )

#Creating inputcols and outputcol
assembler=VectorAssembler(inputCols=['GDP','BMI','Income composition of resources', \
                                     'Schooling','Life expectancy','Total expenditure', \
                                     'percentage expenditure'],outputCol='features')
output=assembler.transform(class_df)
output.select('features','IsDeveloped').show(5)

#changing the value of True or False to 1 and 0
import pyspark.sql.functions as f
final_data=output.select('features','IsDeveloped')
final_data = final_data.withColumn(
        "IsDeveloped",
        f.when(
            f.col("IsDeveloped") == 'False', #Changing the False to 0
            0
        ).when(
            f.col("IsDeveloped") == 'True',  #Changing the True to 1
            1
        )
    )
#To check whether anydata is missing
final_data.count()

#Randomily splitting data
train_data_classifier,test_data__classifier=final_data.randomSplit([0.7,0.3])
#train data
train_data_classifier.describe().show()
#test data
test_data__classifier.describe().show()
train_data_classifier.count()

test_data__classifier.count()

from pyspark.ml.classification import DecisionTreeClassifier
```

```
classifier = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'IsDeveloped',
maxDepth=3)
classifier_Model = classifier.fit(train_data_classifier)

predictions = classifier_Model.transform(test_data__classifier)
predictions.show()

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="IsDeveloped",
predictionCol="prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
assembler=VectorAssembler(inputCols=['GDP','BMI','Income composition of resources', \
                                     'Schooling','Life expectancy','Total expenditure', \
                                     'percentage expenditure'],outputCol='features')
output=assembler.transform(class_df)
output.select('features','IsDeveloped').show(5)
```

```
+--------------------+-----------+
|            features|IsDeveloped|
+--------------------+-----------+
|[584.259216308593...|      false|
|[612.696533203125...|      false|
|[631.744995117187...|      false|
|[669.958984375,17...|      false|
|[63.5372314453125...|      false|
+--------------------+-----------+
only showing top 5 rows
```

```
import pyspark.sql.functions as f
final_data=output.select('features','IsDeveloped')
final_data = final_data.withColumn(
        "IsDeveloped",
        f.when(
            f.col("IsDeveloped") == 'False', #Changing the False to 0
            0
        ).when(
            f.col("IsDeveloped") == 'True',  #Changing the True to 1
            1
        )
    )
final_data.count() #To check whether anydata is missing
```

```
2195
```

```
train_data_classifier,test_data__classifier=final_data.randomSplit([0.7,0.3])

train_data_classifier.describe().show()
test_data__classifier.describe().show()
```

```
+-------+-------------------+
|summary|        IsDeveloped|
+-------+-------------------+
|  count|               1502|
|   mean|0.19973368841544606|
| stddev|0.39993328347291196|
|    min|                  0|
|    max|                  1|
+-------+-------------------+

+-------+-------------------+
|summary|        IsDeveloped|
+-------+-------------------+
|  count|                693|
|   mean|0.17316017316017315|
| stddev|0.37865898765480527|
|    min|                  0|
|    max|                  1|
```

```
train_data_classifier.count()
```

```
1538
```

```
test_data__classifier.count()
```

```
657
```

```
predictions = classifier_Model.transform(test_data__classifier)
predictions.show()
```

```
+--------------------+-----------+-------------+--------------------+----------+
|            features|IsDeveloped|rawPrediction|         probability|prediction|
+--------------------+-----------+-------------+--------------------+----------+
|[1.68134999275207...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[4.61357450485229...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[11.3367795944213...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[12.2773303985595...|          1| [50.0,111.0]|[0.31055900621118...|       1.0|
|[14.1422681808471...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[14.2144117355346...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[15.5743398666381...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[18.2532100677490...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[19.6867179870605...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[24.8190002441406...|          1| [50.0,111.0]|[0.31055900621118...|       1.0|
|[24.9446525573730...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[25.3837318420410...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[26.1525173187255...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[26.3932933807373...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[27.5648250579834...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[32.2227287292480...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[33.4244995117187...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[33.6674919128418...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[39.2818336486816...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
|[42.7379646301269...|          0|[1099.0,43.0]|[0.96234676007005...|       0.0|
+--------------------+-----------+-------------+--------------------+----------+
only showing top 20 rows
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="IsDeveloped", predictionCol="prediction")
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
Test Area Under ROC 0.924119630710089
```