Github Link | Python Equivalent For No 3

A. Correlation

```
df = read.csv("~/pramudya/matkul/data mining/Life Expectancy Data.csv")
```

We use the above code to read Life Expectancy dataset and load it into R

```
head(df)
```

We use **head(df)** to show us the first six rows from our dataset. By reading through the output, we know that the output briefly informs us that our dataset not only contains numerical data but also categorical data.

Description: df [6 × 22]

	Country	Year	Status	Life.expectancy	Adult.Mortality	infant.deaths	Alcohol	percentage.expenditure	Hepatitis.B
	<chr>	<int>	<chr>	<dbl>	<int>	<int>	<dbl>	<dbl>	<int>
1	Afghanistan	2015	Developing	65.0	263	62	0.01	71.279624	65
2	Afghanistan	2014	Developing	59.9	271	64	0.01	73.523582	62
3	Afghanistan	2013	Developing	59.9	268	66	0.01	73.219243	64
4	Afghanistan	2012	Developing	59.5	272	69	0.01	78.184215	67
5	Afghanistan	2011	Developing	59.2	275	71	0.01	7.097109	68
6	Afghanistan	2010	Developing	58.8	279	74	0.01	79.679367	66

6 rows | 1-10 of 22 columns

```
names(df)
```

We use the above code to show the name of each column in the dataset. This dataset has 22 columns containing information about factors affecting life expectancy in different countries in different years.

```
 [1] "Country"                        "Year"                  "Status"                  "Life.expectancy"          "Adult.Mortality"
 [6] "infant.deaths"                  "Alcohol"               "percentage.expenditure"  "Hepatitis.B"              "Measles"
[11] "BMI"                            "under.five.deaths"     "Polio"                   "Total.expenditure"        "Diphtheria"
[16] "HIV.AIDS"                       "GDP"                   "Population"              "thinness..1.19.years"     "thinness.5.9.years"
[21] "Income.composition.of.resources" "Schooling"
```

```
colSums(is.na(df))
```

We use the above code to see the number of missing values in each column. This dataset has a total of 2563 missing values. The missing values are found in 14 columns, namely life expectancy, adult mortality, alcohol, hepatitis B, BMI, polio, total expenditure, diphteria, gdp, population, thinness.1.19.years, thinness.5.9.years, Income.composition.of.resources, and schooling.

```
                       Country                           Year                         Status                Life.expectancy                Adult.Mortality
                             0                              0                              0                             10                             10
                  infant.deaths                        Alcohol        percentage.expenditure                    Hepatitis.B                       Measles
                             0                            194                              0                            553                              0
                           BMI                  under.five.deaths                         Polio              Total.expenditure                     Diphtheria
                            34                              0                             19                            226                             19
                      HIV.AIDS                            GDP                     Population       thinness..1.19.years        thinness.5.9.years
                             0                            448                            652                             34                             34
Income.composition.of.resources                     Schooling
                           167                            163
```

```
df_numerical = df[, c("Life.expectancy", "Year", "Adult.Mortality",
"infant.deaths", "Alcohol", "percentage.expenditure", "Hepatitis.B",
"Measles", "BMI", "under.five.deaths", "Total.expenditure",
"Diphtheria", "HIV.AIDS", "GDP", "Population", "thinness..1.19.years",
"thinness.5.9.years", "Income.composition.of.resources", "Schooling")]

cor(df_numerical)
```

we use the above code to create a new data frame called df_numerical which only includes the columns with numerical data from the original data frame df. Then we use **cor(df_numerical)** to calculate the correlation matrix of the numerical variables in the data frame df_numerical. The resulting correlation matrix shows the pairwise correlations between all numerical variables in the data frame.

	Life.expectancy	Year	Adult.Mortality	infant.deaths	Alcohol	percentage.expenditure	Hepatitis.B	Measles	BMI	under.five.death
Life.expectancy	1	NA	NA	NA	NA	NA	NA	NA	NA	N
Year	NA	1.00000000	NA	-0.03741493	NA	0.03139998	NA	-0.08249298	NA	-0.0429369
Adult.Mortality	NA	NA	1	NA	NA	NA	NA	NA	NA	N
infant.deaths	NA	-0.03741493	NA	1.00000000	NA	-0.08561222	NA	0.50112834	NA	0.9966288
Alcohol	NA	NA	NA	NA	1	NA	NA	NA	NA	N
percentage.expenditure	NA	0.03139998	NA	-0.08561222	NA	1.00000000	NA	-0.05659568	NA	-0.0878523
Hepatitis.B	NA	NA	NA	NA	NA	NA	1	NA	NA	N
Measles	NA	-0.08249298	NA	0.50112834	NA	-0.05659568	NA	1.00000000	NA	0.5078087
BMI	NA	NA	NA	NA	NA	NA	NA	NA	1	N
under.five.deaths	NA	-0.04293699	NA	0.99662888	NA	-0.08785231	NA	0.50780871	NA	1.0000000
Total.expenditure	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
Diphtheria	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
HIV.AIDS	NA	-0.13974136	NA	0.02523132	NA	-0.09785682	NA	0.03089872	NA	0.0380615
GDP	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
Population	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
thinness..1.19.years	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
thinness.5.9.years	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
Income.composition.of.resources	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
Schooling	NA	NA	NA	NA	NA	NA	NA	NA	NA	N

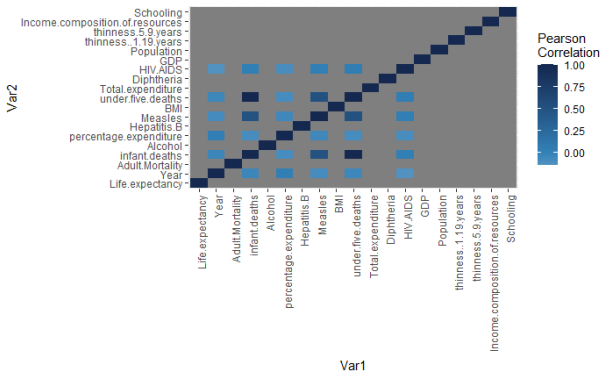
	Total.expenditure	Diphtheria	HIV.AIDS	GDP	Population	thinness..1.19.years	thinness.5.9.years	Income.composition.of.resour
Life.expectancy	NA	NA	NA	NA	NA	NA	NA	
Year	NA	NA	-0.13974136	NA	NA	NA	NA	
Adult.Mortality	NA	NA	NA	NA	NA	NA	NA	
infant.deaths	NA	NA	0.02523132	NA	NA	NA	NA	
Alcohol	NA	NA	NA	NA	NA	NA	NA	
percentage.expenditure	NA	NA	-0.09785682	NA	NA	NA	NA	
Hepatitis.B	NA	NA	NA	NA	NA	NA	NA	
Measles	NA	NA	0.03089872	NA	NA	NA	NA	
BMI	NA	NA	NA	NA	NA	NA	NA	
under.five.deaths	NA	NA	0.03806151	NA	NA	NA	NA	
Total.expenditure	1	NA	NA	NA	NA	NA	NA	
Diphtheria	NA	1	NA	NA	NA	NA	NA	
HIV.AIDS	NA	NA	1.00000000	NA	NA	NA	NA	
GDP	NA	NA	NA	1	NA	NA	NA	
Population	NA	NA	NA	NA	1	NA	NA	
thinness..1.19.years	NA	NA	NA	NA	NA	1	NA	
thinness.5.9.years	NA	NA	NA	NA	NA	NA	1	
Income.composition.of.resources	NA	NA	NA	NA	NA	NA	NA	
Schooling	NA	NA	NA	NA	NA	NA	NA	

One of the variables that has a high correlation is the infant deaths variable with the under five deaths variable with a correlation value of 0.99662888. However, the resulting output is difficult to read because there are too many variables so we will create a visualization using a heat map.

```
library(reshape2)
library(ggplot2)

corr_melted <- melt(cor(df_numerical))
ggplot(corr_melted, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "#DEFCF9", mid = "#3282B8", high =
"#142850",
                       name = "Pearson\nCorrelation") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.7, hjust = 1))
```

We used the code above to visualize the pearson correlation coefficients between all pairs of variables in the **df_numerical** data frame as a heatmap. **cor** function is used to calculate the correlation matrix, **melt** function is used to transform the matrix into a format that can be plotted as tiles.



The grey colour in the output occurs due to the presence of missing values in some of the variables used in the correlation calculation. Therefore, it is necessary to handle these missing values before performing the correlation analysis.

```
df_num <- df[, sapply(df, is.numeric)]
colMeans(df_num, na.rm = TRUE)
```

sapply() is used to apply the **is.numeric()** function to each column of the df data frame, which returns a logical vector used for create a subset data frame that contains only numerical columns. The **colMeans()** is used to calculate the mean value of each column in **df_num** and ignoring any missing values represented by NA.

Year	Life.expectancy	Adult.Mortality	infant.deaths	Alcoho
2.007519e+03	6.922493e+01	1.647964e+02	3.030395e+01	4.602861e+0
percentage.expenditure	Hepatitis.B	Measles	BMI	under.five.death
7.382513e+02	8.094046e+01	2.419592e+03	3.832125e+01	4.203574e+0
Polio	Total.expenditure	Diphtheria	HIV.AIDS	GD
8.255019e+01	5.938190e+00	8.232408e+01	1.742103e+00	7.483158e+0
Population	thinness..1.19.years	thinness.5.9.years	Income.composition.of.resources	Schoolin
1.275338e+07	4.839704e+00	4.870317e+00	6.275511e-01	1.199279e+0

```
for(i in names(df_num)) {
  df_num[, i][is.na(df_num[, i])] = mean(df_num[, i], na.rm=TRUE)
}
colSums(is.na(df_num))
```

We use the above code to replace missing values (NA) in every columns of **df_num** with the mean of each column. **for** loop iterates over each column and then checks if there are any NA values in that column using **is.na**. If there are, it replaces the NA values with the mean of the column. Finally, we confirm that there are no longer any NA values using **colSums(is.na(df_num))**

Year	Life.expectancy	Adult.Mortality	infant.deaths	Alcoho
0	0	0	0	
percentage.expenditure	Hepatitis.B	Measles	BMI	under.five.deaths
0	0	0	0	
Polio	Total.expenditure	Diphtheria	HIV.AIDS	GDP
0	0	0	0	
Population	thinness..1.19.years	thinness.5.9.years	Income.composition.of.resources	Schoolin
0	0	0	0	

```
print(cor(df_num))
```

We use the above code to calculate the correlation matrix of the numerical variables in the data frame **df_num**. The resulting correlation matrix shows the pairwise correlations between all numerical variables in the data frame.

	Year	Life.expectancy	Adult.Mortality	infant.deaths	Alcohol	percentage.expenditure	Hepatitis.B	Measles	BMI	under.five.deat
Year	1.00000000	0.1696226	-0.07886077	-0.03741493	-0.04816759	0.03139998	0.08939827	-0.08249298	0.1083267	-0.042936
Life.expectancy	0.16962259	1.0000000	-0.69635931	-0.19653500	0.39159834	0.38179117	0.20377144	-0.15757382	0.5592553	-0.222503
Adult.Mortality	-0.07886077	-0.6963593	1.00000000	0.07874713	-0.19040781	-0.24281353	-0.13859091	0.03117404	-0.3814494	0.094135
infant.deaths	-0.03741493	-0.1965350	0.07874713	1.00000000	-0.11381227	-0.08561222	-0.17878339	0.50112834	-0.2272200	0.996628
Alcohol	-0.04816759	0.3915983	-0.19040781	-0.11381227	1.00000000	0.33963429	0.07544715	-0.05105499	0.3180703	-0.110777
percentage.expenditure	0.03139998	0.3817912	-0.24281353	-0.08561222	0.33963429	1.00000000	0.01167932	-0.05659568	0.2285372	-0.087852
Hepatitis.B	0.08939827	0.2037714	-0.13859091	-0.17878339	0.07544715	0.01167932	1.00000000	-0.09031694	0.1349285	-0.184412
Measles	-0.08249298	-0.1575738	0.03117404	0.50112834	-0.05105499	-0.05659568	-0.09031694	1.00000000	-0.1759253	0.507808
BMI	0.10832670	0.5592553	-0.38144941	-0.22721997	0.31807030	0.22853723	0.13492851	-0.17592529	1.0000000	-0.237585
under.five.deaths	-0.04293699	-0.2225030	0.09413509	0.99662888	-0.11077713	-0.08785231	-0.18441262	0.50780871	-0.2375859	1.000000
Polio	0.09381958	0.4615738	-0.27269358	-0.17067376	0.21374404	0.14720343	0.40851923	-0.13614628	0.2821559	-0.188703
Total.expenditure	0.08186014	0.2079806	-0.11087454	-0.12656412	0.29489812	0.17341415	0.05008430	-0.10456872	0.2318144	-0.128269
Diphtheria	0.13385337	0.4754184	-0.27301389	-0.17515631	0.21524194	0.14356978	0.49995767	-0.14186137	0.2810588	-0.195650
HIV.AIDS	-0.13974136	-0.5564568	0.52372692	0.02523132	-0.04864971	-0.09785682	-0.10240544	0.03089872	-0.2435476	0.038061
GDP	0.09335084	0.4304930	-0.27705292	-0.10710904	0.31859116	0.88814032	0.06231758	-0.06805959	0.2766447	-0.110640
Population	0.01495063	-0.0196573	-0.01250145	0.54852167	-0.03076466	-0.02464822	-0.10981088	0.23624988	-0.0632376	0.535864
thinness..1.19.years	-0.04759213	-0.4721619	0.29986267	0.46559015	-0.41694557	-0.25119000	-0.10514361	0.22474217	-0.5320247	0.467626
thinness.5.9.years	-0.05062670	-0.4666292	0.30536641	0.47122795	-0.40588068	-0.25272486	-0.10833424	0.22100716	-0.5389106	0.472098
Income.composition.of.resources	0.23633317	0.6924828	-0.44006205	-0.14366278	0.41609923	0.38037355	0.15099204	-0.11576407	0.4798374	-0.161533
Schooling	0.20347119	0.7150663	-0.43510845	-0.19175731	0.49754628	0.38810498	0.17175483	-0.12260854	0.5081055	-0.207111

	Polio	Total.expenditure	Diphtheria	HIV.AIDS	GDP	Population	thinness..1.19.years	thinness.5.9.years
Year	0.09381958	0.08186014	0.13385337	-0.13974136	0.09335084	0.014950626	-0.04759213	-0.0506267
Life.expectancy	0.46157378	0.20798062	0.47541838	-0.55645682	0.43049302	-0.019637702	-0.47216188	-0.4666292
Adult.Mortality	-0.27269358	-0.11087454	-0.27301389	0.52372692	-0.27705292	-0.012501453	0.29986267	0.3053664
infant.deaths	-0.17067376	-0.12656412	-0.17515631	0.02523132	-0.10710904	0.548521669	0.46559015	0.4712279
Alcohol	0.21374404	0.29489812	0.21524194	-0.04864971	0.31859116	-0.030764657	-0.41694557	-0.4058807
percentage.expenditure	0.14720343	0.17341415	0.14356978	-0.09785682	0.88814032	-0.024648218	-0.25119000	-0.2527249
Hepatitis.B	0.40851923	0.05008430	0.49995767	-0.10240544	0.06231758	-0.109810878	-0.10514361	-0.1083342
Measles	-0.13614628	-0.10456872	-0.14186137	0.03089872	-0.06805959	0.236249877	0.22474217	0.2210072
BMI	0.28215592	0.23181440	0.28105881	-0.24354758	0.27664474	-0.063237604	-0.53202475	-0.5389106
under.five.deaths	-0.18870311	-0.12826939	-0.19565055	0.03806151	-0.11064032	0.535864022	0.46762640	0.4720986
Polio	1.00000000	0.13012949	0.67355332	-0.15948864	0.19398031	-0.034881799	-0.21993756	-0.2207101
Total.expenditure	0.13012949	1.00000000	0.14559660	-0.00138272	0.12146709	-0.066698201	-0.26872376	-0.2752400
Diphtheria	0.67355332	0.14559660	1.00000000	-0.16478684	0.18279456	-0.025457689	-0.22781986	-0.2211053
HIV.AIDS	-0.15948864	-0.00138272	-0.16478684	1.00000000	-0.13451379	-0.027318431	0.20392213	0.2071396
GDP	0.19398031	0.12146709	0.18279456	-0.13451379	1.00000000	-0.025611731	-0.26774463	-0.2723995
Population	-0.03488180	-0.06669820	-0.02545769	-0.02731843	-0.02561173	1.000000000	0.23611740	0.2339409
thinness..1.19.years	-0.21993756	-0.26872376	-0.22781986	0.20392213	-0.26774463	0.236117395	1.00000000	0.9391020
thinness.5.9.years	-0.22071010	-0.27524000	-0.22110531	0.20713956	-0.27239955	0.233940899	0.93910199	1.0000000
Income.composition.of.resources	0.35539764	0.14909473	0.37172915	-0.24745353	0.44031708	-0.007951445	-0.40666173	-0.3957785
Schooling	0.38583163	0.21831013	0.38994380	-0.21861975	0.42948916	-0.029464903	-0.44614009	-0.4357772

	Income.composition.of.resources	Schooling
Year	0.236333172	0.2034712
Life.expectancy	0.692482805	0.7150663
Adult.Mortality	-0.440062048	-0.4351085
infant.deaths	-0.143662780	-0.1917573
Alcohol	0.416099225	0.4975463
percentage.expenditure	0.380373551	0.3881050
Hepatitis.B	0.150992044	0.1717548
Measles	-0.115764074	-0.1226085
BMI	0.479837369	0.5081055
under.five.deaths	-0.161533413	-0.2071114
Polio	0.355397638	0.3858316
Total.expenditure	0.149094726	0.2183101
Diphtheria	0.371729147	0.3899438
HIV.AIDS	-0.247453534	-0.2186198
GDP	0.440317075	0.4294892
Population	-0.007951445	-0.0294649
thinness..1.19.years	-0.406661733	-0.4461401
thinness.5.9.years	-0.395778501	-0.4357772
Income.composition.of.resources	1.000000000	0.7962071
Schooling	0.796207053	1.0000000

As we can see in the output, there are no NA values because we have handled the missing values. Based on the output, we can also see that life expectancy has a strong positive correlation with Income composition of resources and Schooling. It also has moderate positive correlations with Alcohol, BMI, Diphtheria, GDP, Percentage expenditure, and Polio. On the other hand, life expectancy has strong negative correlations with HIV/AIDS, and moderate negative correlations with Adult Mortality, Thinness 1-19 years, and Thinness 5-9 years. Infant deaths and under-five deaths are negatively correlated with life expectancy, but to a lesser extent. It is important to note that correlation does not imply causation and further analysis is needed to draw any causal relationships.

```
library(dplyr)

corr_melted <- melt(cor(df_num))
corr_melted_sorted <- corr_melted %>% arrange(desc(value))
head(corr_melted_sorted, 40)
```

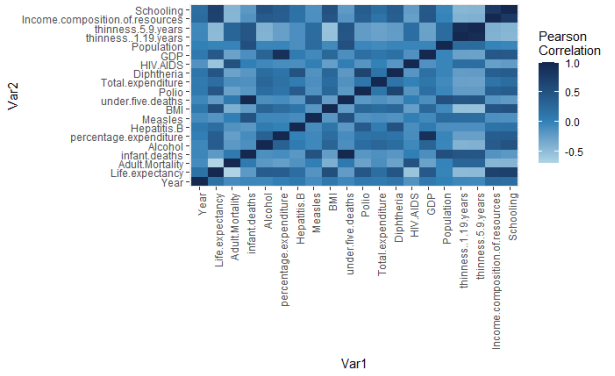
We used the code above to sorts the correlation values between all pairs of numeric variables in the dataset in descending order and print the top 40 pairs.

	Var1 <fctr>	Var2 <fctr>	value <dbl>
1	Year	Year	1.0000000
2	Life.expectancy	Life.expectancy	1.0000000
3	Adult.Mortality	Adult.Mortality	1.0000000
4	infant.deaths	infant.deaths	1.0000000
5	Alcohol	Alcohol	1.0000000
6	percentage.expenditure	percentage.expenditure	1.0000000
7	Hepatitis.B	Hepatitis.B	1.0000000
8	Measles	Measles	1.0000000
9	BMI	BMI	1.0000000
10	under.five.deaths	under.five.deaths	1.0000000
11	Polio	Polio	1.0000000
12	Total.expenditure	Total.expenditure	1.0000000
13	Diphtheria	Diphtheria	1.0000000
14	HIV.AIDS	HIV.AIDS	1.0000000
15	GDP	GDP	1.0000000
16	Population	Population	1.0000000
17	thinness..1.19.years	thinness..1.19.years	1.0000000
18	thinness.5.9.years	thinness.5.9.years	1.0000000
19	Income.composition.of.resources	Income.composition.of.resources	1.0000000
20	Schooling	Schooling	1.0000000
21	under.five.deaths	infant.deaths	0.9966289
22	infant.deaths	under.five.deaths	0.9966289
23	thinness.5.9.years	thinness..1.19.years	0.9391020
24	thinness..1.19.years	thinness.5.9.years	0.9391020
25	GDP	percentage.expenditure	0.8881403
26	percentage.expenditure	GDP	0.8881403
27	Schooling	Income.composition.of.resources	0.7962071
28	Income.composition.of.resources	Schooling	0.7962071
29	Schooling	Life.expectancy	0.7150663
30	Life.expectancy	Schooling	0.7150663
31	Income.composition.of.resources	Life.expectancy	0.6924828
32	Life.expectancy	Income.composition.of.resources	0.6924828
33	Diphtheria	Polio	0.6735533
34	Polio	Diphtheria	0.6735533
35	BMI	Life.expectancy	0.5592553
36	Life.expectancy	BMI	0.5592553
37	Population	infant.deaths	0.5485217
38	infant.deaths	Population	0.5485217
39	Population	under.five.deaths	0.5358640
40	under.five.deaths	Population	0.5358640

The output reveals that under.five.deaths and infant.deaths have a high correlation value of 0.9966, indicating a strong positive relationship between the two variables. Similarly, thinness.5.9.years and thinness..1.19.years are highly correlated at 0.9391. Additionally, the table shows that there are several pairs of variables that have relatively high correlation values with Life.expectancy, such as Schooling, Income.composition.of.resources, BMI, and GDP

```
ggplot(corr_melted, aes(x = Var1, y = Var2, fill = value)) + geom_tile()
+
scale_fill_gradient2(low = "#DEFCF9", mid = "#3282B8", high = "#142850",
name = "Pearson\nCorrelation") + theme(axis.text.x = element_text(angle
= 90, vjust = 0.7, hjust = 1))
```

We used the code above to visualize the pearson correlation coefficients between all pairs of variables in the **df_num.** The darker colors indicate a strong positive correlation, meaning that when one variable increases, the other variable also tends to increase. On the other hand, the lighter colors indicate a strong negative correlation, indicating that when one variable increases, the other variable tends to decrease.



Looking at the results above, we can tell that we've successfully handled all missing values because there's no longer any grey colour in the output. Visually, we can see that there are quite a lot of highly correlated variables such as thinness.5.9.years with thinness..1.19.years, Schooling along with Income.composition.of.resources with life expectancy, infant.deaths with under.five.deaths, percentage.expenditure with GDP, Adult.Mortality as well as HIV/AIDS with life expectancy, and thinness..1.19.years together with thinness.5.9.years with BMI.

B. Statistical Testing

In this section, we will employ various statistical tests designed to address our specific research question or hypothesis. By leveraging the power of statistical analysis, we aim to gain deeper insights and draw meaningful conclusions from the data at hand.

1. How is the distribution of life expectancy in this dataset divided into specific quantiles?

```
df_stat <- df_num
quantiles <- quantile(df_stat$Life.expectancy, probs = seq(0, 1, 0.25))
quantiles
```

We use this code to calculate the quantiles of the **Life.expectancy** variable. First, we make a copy from **df_num** to **df_stat** to ensure that none of the contents of our dataset will change. After that we calculate the quantiles using the **quantile()** function with **seq(0, 1, 0.25)** in order to get the quantiles at 0%, 25%, 50%, 75%, and 100% percentages and then we save the calculation results to the **quantiles** variable then we display the results. By using this code we will be able to find out how the distribution of the Life.expectancy variable in our dataset.

```
  0%  25%  50%  75% 100%
36.3 63.2 72.0 75.6 89.0
```

Based on the output, we can see that at the 25% quantile, the life expectancy value is 63.2. This means that 25% of the population in the dataset has a life expectancy below this value. At the 50% quantile, the life expectancy value is 72.0. This means that half the population in the dataset has a life expectancy below this value. At the 75% quantile the life expectancy value is 75.6. This means that 75% of the population in the dataset has a life expectancy below this value. At the 100% quantile the life expectancy value is 89.0, which means that 89 years is the maximum life expectancy in this dataset.

2. Is the minimum value of the life expectancy variable is in a normal distribution with the mean and standard deviation of the variable?

```
minLE <- min(df_stat$Life.expectancy)
maxLE <- max(df_stat$Life.expectancy)
meanLE <- mean(df_stat$Life.expectancy)
sdLE <- sd(df_stat$Life.expectancy)
pMin <- pnorm(q = minLE, mean = meanLE, sd = sdLE)
pMin
```

We use the above code to determine the extent to which the minimum value of the life expectancy variable is within a normal distribution using the mean and standard deviation of that variable itself. The result of this code is **pMin**, which represents the probability of the minimum value being within a normal distribution. The higher the pMin value, the higher the probability that the minimum value of the variable comes from a normal distribution.

Initially, we calculate the minimum value of the life expectancy variable using the **min()** function, then we calculate its maximum value using the **max()** function. Next, we calculate the mean using the **mean()** function and the standard deviation using the **sd()** function. Finally, we use the **pnorm()** function with the arguments **q = minLE, mean = meanLE, and sd = sdLE** to calculate the probability (pMin) of the minimum value being within a normal distribution.

```
[1] 0.0002670969
```

Based on the output produced, the pMin value is very low, indicating that the possibility of the minimum value of the life expectancy variable coming from a normal distribution is very low.

3. What are the quantile values of a normal distribution with specified mean and standard deviation, for ten different probabilities, ranging from 0 to 1 with intervals of 0.1?

```
qnorm(mean = meanLE, sd = sdLE, p = seq(0, 1, 0.1))
```

We use the above code to generate quantile values for the life expectancy variable from a normal distribution with the pre-determined mean and standard deviation. The output of this code will provide the quantile values corresponding to the specified probabilities. The code utilizes the **qnorm()** function with the argument **p = seq(0, 1, 0.1)** to determine ten different probabilities ranging from 0 to 1 with an interval of 0.1.

```
 [1]     -Inf 57.04040 61.22310 64.23912 66.81620 69.22493 71.63366 74.21074 77.22676 81.40946      Inf
```

Based on the code above, the 0% quantile (-Inf) means that -inf is the lowest value that can be obtained in a normal distribution. The 10% quantile (57.04040) means that about 10% of the data falls below this value. The 20% quantile (61.22310) means that about 20% of the data is below this value. 30% Quantile (64.23912) means that about 30% of the data is below this value. The 40% quantile (66.81620) means that about 40% of the data is below this value. The 50% quantile (69.22493) means that about 50% of the data is below this value. The 60% quantile (71.63366) means that about 60% of the data is below this value. The 70% quantile (74.21074) means that about 70% of the data is below this value. The 80% quantile (77.22676) means that about 80% of the data is below this value. The 90% quantile (81.40946) means that about 90% of the data is below this value. The 100% quantile (Inf) means

that inf is the highest value that can be obtained. In short, these values are an illustration of how the normal distribution value for the life expectancy variable based on the mean and standard deviation that we have determined.

4. Is there a significant difference between the average life expectancy in this dataset and the specified population mean value (70 years)?

```
t.test(df_stat$Life.expectancy, mu = 70, alternative = "two.sided")
```

Based on the output of questions 1, 2, and 3, we use the above code to evaluate whether there is a significant difference between the average life expectancy in the dataset and the specified population mean value (70 years). In this case, the null hypothesis (H0) is that the average life expectancy in the dataset is equal to 70 years, while the alternative hypothesis (H1) is that the average life expectancy in the dataset is significantly different from 70 years.

```
        One Sample t-test

data:  df_stat$Life.expectancy
t = -4.4187, df = 2937, p-value = 1.029e-05
alternative hypothesis: true mean is not equal to 70
95 percent confidence interval:
 68.88100 69.56886
sample estimates:
mean of x
 69.22493
```

From the results, we can infer that there is a significant difference between the average ife expectancy in the dataset and the specified population mean value (70 years). The low p-value (1.029e-05) suggests that the difference is not by chance and holds significance. The 95% confidence interval indicates that the average life expectancy falls between 68.88100 and 69.56886. Hence, we can **reject the null hypothesis (H0)** that the average life expectancy is equal to 70 years.

C. Predictive Modelling

We'll be using Support Vector Regression (SVR) to create our predictive models.

```
Max <- apply(df_num, 2, max)
Min <- apply(df_num, 2, min)
Max
```

We use the above code to calculate the maximum and minimum values for each column in **df_num.** The **"apply"** function is used to apply a function to each column, **"2"** is used to specifies that we want to apply the function to each column not row, and **"max"** or **"min"** returns the maximum or minimum value of each column.

Max is used to show all the max value in each column of **df_num**

```
                      Year                Life.expectancy                Adult.Mortality                   infant.deaths                         Alcohol
              2.015000e+03                  8.900000e+01                  7.230000e+02                   1.800000e+03                   1.787000e+01
percentage.expenditure                  Hepatitis.B                       Measles                            BMI                  under.five.deaths
              1.947991e+04                  9.900000e+01                  2.121830e+05                   8.730000e+01                   2.500000e+03
                     Polio            Total.expenditure                    Diphtheria                       HIV.AIDS                            GDP
              9.900000e+01                  1.760000e+01                  9.900000e+01                   5.060000e+01                   1.191727e+05
                Population         thinness..1.19.years         thinness.5.9.years Income.composition.of.resources                      Schooling
              1.293859e+09                  2.770000e+01                  2.860000e+01                   9.480000e-01                   2.070000e+01
```

```
Min
```

We use the above code show all the min value in each column of **df_num**

Year	Life.expectancy	Adult.Mortality	infant.deaths	Alcoho
2000.00000	36.30000	1.00000	0.00000	0.0100
percentage.expenditure	Hepatitis.B	Measles	BMI	under.five.death
0.00000	1.00000	0.00000	1.00000	0.0000
Polio	Total.expenditure	Diphtheria	HIV.AIDS	GD
3.00000	0.37000	2.00000	0.10000	1.6813
Population	thinness..1.19.years	thinness.5.9.years	Income.composition.of.resources	Schoolin
34.00000	0.10000	0.10000	0.00000	0.0000

```
scaled <- as.data.frame(scale(df_num, center = Min, scale = Max - Min))
head(scaled)
```

We use the above code to calculate the normalized data scale value by entering the minimum and maximum values of each column. The code is used to rescale the numerical values in the dataset between the values of 0 and 1. Normalizing the data is important because it ensures that the model does not give more weight to the variables with larger scales, which could skew the results.

	Year	Life.expectancy	Adult.Mortality	infant.deaths	Alcohol	percentage.expenditure	Hepatitis.B	Measles	BI
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1.0000000	0.5445920	0.3628809	0.03444444	0	0.0036591349	0.6530612	0.005438701	0.209733
2	0.9333333	0.4478178	0.3739612	0.03555556	0	0.0037743283	0.6224490	0.002318753	0.203939
3	0.8666667	0.4478178	0.3698061	0.03666667	0	0.0037587051	0.6428571	0.002026553	0.198146
4	0.8000000	0.4402277	0.3753463	0.03833333	0	0.0040135816	0.6734694	0.013134888	0.192352
5	0.7333333	0.4345351	0.3795014	0.03944444	0	0.0003643296	0.6836735	0.014200007	0.187717
6	0.6666667	0.4269450	0.3850416	0.04111111	0	0.0040903352	0.6632653	0.009373984	0.181923

By looking at the output of **head(scaled)** we can see that all values in each column have been normalized to values between 0 and 1.

```
library(caTools)
set.seed(42)
split = sample.split(scaled$Life.expectancy, SplitRatio = 0.80) #train
80%, test 20%

train <- subset(scaled, split == TRUE)
test <- subset(scaled, split == FALSE)
```

The code above is used to split our data into two sets - a training set and a testing set, with 80% of the data allocated for training and 20% for testing. Setting a seed in the code ensures that the same random split will be used whenever the code is run in the future, which helps with reproducibility.

```
library(e1071)

modelSVR <- svm(f, data = train, type = "eps-regression", kernel =
"radial")
summary(modelSVR)
```

we use the code above to create a predictive model using SVR. In this code we set the type of SVM to "eps-regression" because we want to **focus on regression rather than classification**. In this code we also use "radial" kernel which is suitable for regression tasks. The code **summary(modelSVR)** is used to obtain a summary of the SVR model's relevant statistics.

```
Call:
svm(formula = f, data = train, type = "eps-regression", kernel = "radial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.05263158
    epsilon:  0.1


Number of Support Vectors:  1341
```

Based on the output we can see that the model was trained with a cost parameter of 1 and a gamma parameter of 0.05263158. The epsilon value, which represents the margin of error allowed in the regression, was set to 0.1 and The number of support vectors used by the model is 1341.

```
predictedSVR <- predict(modelSVR, newdata = test)
resultsSVR <- cbind(predictedSVR,test$Life.expectancy)
colnames(resultsSVR) <- c('Predicted Value','Ground Truth')
resultsSVR <- as.data.frame(resultsSVR)
resultsSVR
```

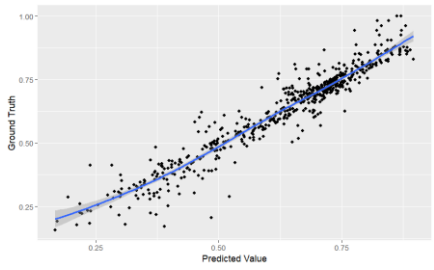
We used the code above to make predictions on the test dataset using the SVR model. The predicted values are then combined with the actual values from the test dataset to create a data frame **resultsSVR** so we can compare the predicted values against the actual values.

	Predicted Value <dbl>	Ground Truth <dbl>
1	0.4589726	0.5445920
9	0.4360704	0.4022770
12	0.4704445	0.3927894
16	0.3018947	0.3510436
17	0.7879666	0.7874763
20	0.7712364	0.7703985
30	0.7129691	0.7020873
34	0.7510212	0.7419355
45	0.6808974	0.6717268
50	0.3314092	0.2922201

By just looking at the first 10 rows we can see that our model can predict the life expectancy values good enough, now we will try to see the visualization between the actual and predicted values.

```
plotSVR <- ggplot(resultsSVR,aes(x=resultsSVR$`Predicted
Value`,y=resultsSVR$`Ground Truth`)) + geom_point() + stat_smooth() +
xlab("Predicted Value") + ylab("Ground Truth")
plotSVR
```

We used the code above to create a scatter plot that shows the relationship between the predicted life expectancy values and the actual values. The x-axis represents the predicted values, and the y-axis represents the actual values.



By just looking at the plot we know that our model is somehow good but not that good to predict the life expectancy, because if the points are tightly clustered around the line, the model's performance is good. If the points are scattered widely around the line, the model's performance may be poor. Now we will check the performance of our model by calculate the MSE.

```
MSE_SVR <- sum((resultsSVR$`Predicted Value` - resultsSVR$`Ground
Truth`)^2)/nrow(test)
MSE_SVR
```

We use the code above calculate the mean squared error (MSE) of the Support Vector Regression model. The code calculates the MSE by subtracting the predicted

values from the actual values, squaring the difference, summing up all the squared differences, and then dividing the sum by the number of rows in the test dataset.

```
[1] 0.002549063
```

The results show that the SVR model performs good in predicting life expectancy for the test set. The average error of the Support Vector Regression model is so small which is only 0.002549063 squared units. This suggests that the **SVR model is a good fit** for the data and provides accurate predictions of life expectancy values.

Now that we have completed building the model in R, it's time to create the Python equivalent for the same model.

Python Equivalent

```
import os
import random
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

In this section we will first import the libraries and functions that we will use. Briefly, the following is an explanation of each function and library that we import:

- **os:** used to set a seed in python
- **random:** used to set a seed
- **numpy:** used for matrix things
- **pandas:** used for data frame operations
- **seaborn:** for data visualization
- **matplotlib:** for data visualization
- **SVR:** for create a SVR model
- **mean_squared_error:** used for calculating MSE
- **train_test_split:** splits data into train and test data

```
df = pd.read_csv('/content/Life Expectancy Data.csv')
```

We use the above code to read Life Expectancy dataset and load it into colab

```
df.head()
```

We use **df.head()** to show us the first five rows from our dataset. By reading through the output, we know that the output briefly informs us that our dataset not only contains numerical data but also categorical data.

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1

5 rows × 22 columns

```
df_num = df.select_dtypes(include=['number'])
df_num.head()
```

The code above is used to create a subset of the original data frame (**df**) that includes only the numerical columns. It then checks if **df_num** contains only numerical columns by printing out the result.

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Pop
0	2015	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1	83	6.0	8.16	65.0	0.1	584.259210	337
1	2014	59.9	271.0	64	0.01	73.523582	62.0	492	18.6	86	58.0	8.18	62.0	0.1	612.696514	3
2	2013	59.9	268.0	66	0.01	73.219243	64.0	430	18.1	89	62.0	8.13	64.0	0.1	631.744976	317
3	2012	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6	93	67.0	8.52	67.0	0.1	669.959000	36
4	2011	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2	97	68.0	7.87	68.0	0.1	63.537231	29

The output of the code confirms that **df_num** indeed contains only numerical columns, indicating that the filtering process was successful.

```
print(df_num.isnull().sum())
```

We used the above code to see the number of missing values in each column. This dataset has a total of 2563 missing values. The missing values are found in 14 columns, namely life expectancy, adult mortality, alcohol, hepatitis B, BMI, polio, total expenditure, diphteria, gdp, population, thinness.1.19.years, thinness.5.9.years, Income.composition.of.resources, and schooling.

```
Year                               0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                              448
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

```
df_num = df_num.fillna(df_num.mean())
```

We used the above code to replace missing values (NA) in every columns of **df_num** with the mean of each column.

```
print(df_num.isnull().sum())
```

We used the code above to confirm that there are no longer any NA values and by looking at the output we can say that the imputation has been done successfully.

```
Year                               0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
GDP                                0
Population                         0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
dtype: int64
```

```
Max = df_num.max()
Min = df_num.min()
```

We used the above code to calculate the maximum and minimum values for each column in **df_num.**

```
scaled = (df_num - Min) / (Max - Min)
```

We used the above code to calculate the normalized data scale value by entering the minimum and maximum values of each column. The code is used to rescale the numerical values in the dataset between the values of 0 and 1.

```
scaled.head()
```

We use the code above to show the first 5 rows of data frame scaled. By looking at the output we can see that all values in each column have been normalized to values between 0 and 1

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	
0	1.000000	0.544592	0.362881	0.034444	0.0	0.003659	0.653061	0.005439	0.209733	0.0332	0.031250	0.452118	0.649485	0.0	0.0
1	0.933333	0.447818	0.373961	0.035556	0.0	0.003774	0.622449	0.002319	0.203940	0.0344	0.572917	0.453279	0.618557	0.0	0.0
2	0.866667	0.447818	0.369806	0.036667	0.0	0.003759	0.642857	0.002027	0.198146	0.0356	0.614583	0.450377	0.639175	0.0	0.0
3	0.800000	0.440228	0.375346	0.038333	0.0	0.004014	0.673469	0.013135	0.192352	0.0372	0.666667	0.473012	0.670103	0.0	0.0
4	0.733333	0.434535	0.379501	0.039444	0.0	0.000364	0.683673	0.014200	0.187717	0.0388	0.677083	0.435287	0.680412	0.0	0.0

```
SEED = 42
os.environ['PYTHONHASHSEED']=str(SEED)
os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
random.seed(SEED)
np.random.seed(SEED)
```

In this section we install seeds for python and numpy operations so that the results we get remain the same, if we want to re-run the code

```
scaled.columns
```

This code is used to display the name of each column in the scaled data frame. We do this so that later when separating the independent and dependent variables we can just copy and paste the column names

```
Index(['Year', '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 years',
       ' thinness 5-9 years', 'Income composition of resources', 'Schooling'],
      dtype='object')
```

```
x = scaled[['Year', '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 years',
       ' thinness 5-9 years', 'Income composition of resources',
'Schooling']]
y = scaled[['Life expectancy ']]
```

We used the code above to when separating the independent and dependent variables.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

We used the code above to split our data into two sets - a training set and a testing set, with 80% of the data allocated for training and 20% for testing.

```
modelSVR = SVR(kernel='rbf', epsilon=0.1)
modelSVR.fit(x_train, y_train)
```

We used the code above to create an SVR model then we train the model with the train set.

```
predictedSVR = modelSVR.predict(x_test)
```

The code above is used to make a prediction of y_test using model that we have already trained before.

```
resultsSVR = pd.DataFrame({'Predicted Value': predictedSVR.flatten(),
'Ground Truth': y_test.values.flatten()})
resultsSVR
```

We used the code above to combine predicted values with the actual values from the test dataset to create a data frame **resultsSVR** so we can compare the predicted values against the actual values.

	Predicted Value	Ground Truth
0	0.636053	0.709677
1	0.801080	0.751423
2	0.755124	0.719165
3	0.749896	0.768501
4	0.294846	0.296015
...	...	...
583	0.473151	0.444023
584	0.791605	0.705882
585	0.520019	0.470588
586	0.643725	0.631879
587	0.709149	0.715370

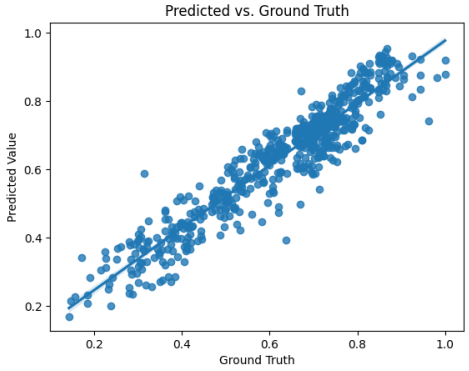
588 rows × 2 columns

By just looking at the first and last 5 rows we can see that our model can predict the life expectancy values good enough, now we will try to see the visualization between the actual and predicted values.

```
predicted = resultsSVR['Predicted Value']
ground_truth = resultsSVR['Ground Truth']

sns.regplot(x=ground_truth, y=predicted)
plt.xlabel('Ground Truth')
plt.ylabel('Predicted Value')
plt.title('Predicted vs. Ground Truth')
plt.show()
```

We use the first and second line of code to extract the predicted and actual value from resultsSVR data frame then we try to plots a scatter plot for predicted value and the ground truth or the actual values.



By just looking at the plot we know that our model is good to predict the life expectancy, because if the points are tightly clustered around the line, the model's performance is good. If the points are scattered widely around the line, the model's performance may be poor. Now we will check the performance of our model by calculate the MSE.

```
mse = mean_squared_error(ground_truth, predicted)
print("Mean Squared Error (MSE):", mse)
```

We use the code above calculate the mean squared error (MSE) of the Support Vector Regression model.

```
Mean Squared Error (MSE): 0.003522840953962061
```

The results show that the SVR model performs good in predicting life expectancy for the test set. The average error of the Support Vector Regression model is so small which is only 0.003522840953962061 squared units. This suggests that the **SVR model is a good fit** for the data and provides accurate predictions of life expectancy values.

Although it is small, we can see the difference in MSE of the models created in R and those created in Python. The MSE of the model created in python has a value of 0.003522840953962061 while in R the MSE value is 0.002549063.

The difference in MSE values between models created in R and Python can occur due to several things, one of which is due to differences in functions and libraries used in Python and R. In addition, even though they use the same algorithm, the implementation of algorithms in Python and R can be different.