Department of Computer Engineering
College of Engineering
Polytechnic University of the Philippines Sta. Mesa

CMPE 40163: Big Data using PySpark Final Requirement
Data Manipulation of Heart Failure Clinical Records Dataset

Submitted by:

Marcial, John Erwin D.
BSCpE 3-2

Submitted to:

EDCEL B. ARTIFICIO

I. Introduction

For this project, I chose to research the field in which I am most interested - medicine, namely **cardiovascular illnesses (CVDs)**. According to the World Health Organization (WHO) cardiovascular disorders are the major cause of death worldwide. In 2019, an estimated 17.9 million individuals died from CVDs, accounting for 32% of all global deaths. Heart attacks and strokes were responsible for 85% of these deaths.

With **heart failure** being a major complication of CVDs, the dataset used in this undertaking contains 12 variables that can be used to predict heart failure mortality. Most cardiovascular diseases can be avoided by addressing behavioral risk factors such as cigarette use, poor diet and obesity, physical inactivity, and hazardous alcohol consumption through population-wide interventions.

People with cardiovascular disease or at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia, or pre-existing disease) require early detection and management, which a machine learning model can greatly assist with.

II. Data Dictionary

The dataset I utilized is Kaggle's **Heart Failure Clinical Records Dataset**, which contains **(299 rows, 13 columns)**. The rows specify the collection of data in relation to the 13 columns shown below.

13 Clinical Features	Data type
age: age of the patient (years)	**Float**
anaemia: decrease of red blood cells or hemoglobin	**Boolean**
high blood pressure: if the patient has hypertension	**Boolean**
creatinine phosphokinase (CPK): level of the CPK enzyme in the blood (mcg/L)	**Integer**
diabetes: if the patient has diabetes	**Boolean**
ejection fraction: percentage of blood leaving the heart at each contraction (%)	**Integer**
platelets: platelets in the blood (kiloplatelets/mL)	**Float**
sex: woman or man	**Boolean**
serum creatinine: level of serum creatinine in the blood (mg/dL)	**Float**
serum sodium: level of serum sodium in the blood (mEq/L)	**Integer**
smoking: if the patient smokes or not	**Boolean**
time: follow-up period (days)	**Integer**
[target] death event: if the patient deceased during the follow-up period	**Boolean**

I've also included screenshots of programs and outputs below that helped me learn and understand more about the dataset I'm working with.

```
[78]
#Counts the number of rows and columns of the dataset
print('Shape of the dataset: ', (file_df.count(), len(file_df.columns)))

Shape of the dataset:  (299, 13)
```

```python
#Shows the schema of the dataframe
file_df.printSchema()
```

```
root
 |-- age: double (nullable = true)
 |-- anaemia: integer (nullable = true)
 |-- creatinine_phosphokinase: integer (nullable = true)
 |-- diabetes: integer (nullable = true)
 |-- ejection_fraction: integer (nullable = true)
 |-- high_blood_pressure: integer (nullable = true)
 |-- platelets: double (nullable = true)
 |-- serum_creatinine: double (nullable = true)
 |-- serum_sodium: integer (nullable = true)
 |-- sex: integer (nullable = true)
 |-- smoking: integer (nullable = true)
 |-- time: integer (nullable = true)
 |-- DEATH_EVENT: integer (nullable = true)
```

```python
#Displays basic statistics of the dataset
file_df.describe().show()
```

```
----------+------------------+
|summary|               age|           anaemia|creatinine_phosphokinase|          diabetes|
ejection_fraction|high_blood_pressure|         platelets|  serum_creatinine|      serum_sodium|               sex|
smoking|              time|        DEATH_EVENT|
+-------+------------------+------------------+------------------------+------------------+------------------+------
------------+------------------+------------------+------------------+------------------+------------------+-------
----------+------------------+
|  count|               299|               299|                     299|               299|               299|
299|               299|               299|               299|               299|               299|
299|                299|
|   mean| 60.83389297658862| 0.431438127090301|       581.8394648829432|0.4180602006688963| 38.08361204013378|
0.3511705685618729|263358.02926421416|
1.393879598662207|136.62541806020067|0.6488294314381271|0.3210702341137124|130.2608695652174| 0.3210702341137124|
| stddev|11.894809074044469|0.4961072681330795|       970.2878807124358|0.4940670651036091|11.834840741039168|
0.4781363790627446|  97804.2368685983|1.0345100640898544|
4.412477283909232|0.4781363790627448|0.4676704280567715|77.61420795029336|0.46767042805677195|
|    min|              40.0|                 0|                      23|                 0|                14|
0|           25100.0|               0.5|               113|                 0|                 0|                 4|
0|
|    max|              95.0|                 1|                    7861|                 1|                80|
1|          850000.0|               9.4|               148|                 1|                 1|               285|
1|
+-------+------------------+------------------+------------------------+------------------+------------------+------
```

III. Data Manipulation using an RDD

- I started with importing the driver programs that I will be needing in creating an RDD, as well as in creating the dataframe for later.

```
[24]
from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.sql import SparkSession

spark:SparkSession = SparkSession.builder.master('local[*]').appName("MrSparkIDontFeelSoGood").getOrCreate()
```

- Loaded my chosen csv file as an RDD and saved it to the variable file_rdd_raw.

```
[23]
#Load dataset as an RDD
file_rdd_raw = spark.sparkContext.textFile("heart_failure_clinical_records_dataset.csv")
```

- Display the data under the file_rdd_raw which will serve as the pre RDD transformation.

```
[3]
#Displays the file_rdd_raw
file_rdd_raw.collect()
```

```
['age,anaemia,creatinine_phosphokinase,diabetes,ejection_fraction,high_blood_pressure,platelets,serum_creatinine,seru
 '75,0,582,0,20,1,265000,1.9,130,1,0,4,1',
 '55,0,7861,0,38,0,263358.03,1.1,136,1,0,6,1',
 '65,0,146,0,20,0,162000,1.3,129,1,1,7,1',
 '50,1,111,0,20,0,210000,1.9,137,1,0,7,1',
 '65,1,160,1,20,0,327000,2.7,116,0,0,8,1',
 '90,1,47,0,40,1,204000,2.1,132,1,1,8,1',
 '75,1,246,0,15,0,127000,1.2,137,1,0,10,1',
 '60,1,315,1,60,0,454000,1.1,131,1,1,10,1',
 '65,0,157,0,65,0,263358.03,1.5,138,0,0,10,1',
 '80,1,123,0,35,1,388000,9.4,133,1,1,10,1',
 '75,1,81,0,38,1,368000,4,131,1,1,10,1',
 '62,0,231,0,25,1,253000,0.9,140,1,1,10,1',
 '45,1,981,0,30,0,136000,1.1,137,1,0,11,1',
 '50,1,168,0,38,1,276000,1.1,137,1,0,11,1',
 '49,1,80,0,30,1,427000,1,138,0,0,12,0',
 '82,1,379,0,50,0,47000,1.3,136,1,0,13,1',
 '87,1,149,0,38,0,262000,0.9,140,1,0,14,1',
 '45,0,582,0,14,0,166000,0.8,127,1,0,14,1',
 '70,1,125,0,25,1,237000,1,140,0,0,15,1',
 '48,1,582,1,55,0,87000,1.9,121,0,0,15,1',
 '65,1,52,0,25,1,276000,1.3,137,0,0,16,0',
 '65,1,128,1,30,1,297000,1.6,136,0,0,20,1',
 '68,1,220,0,35,1,289000,0.9,140,1,1,20,1',
 '53,0,63,1,60,0,368000,0.8,135,1,0,22,0',
 '75,0,582,1,30,1,263358.03,1.83,134,0,0,23,1',
 '80,0,148,1,38,0,149000,1.9,144,1,1,23,1',
```

- The first data manipulation that I have used is the **map transformation**. I separated the lines of the file_rdd_raw by columns using the lambda function and saved it to the file_rdd variable. Using the **collect** action, I was able to display the data from the file_rdd.

```
[27]
#Map and Collect
#Split the lines of file_rdd_raw by columns
file_rdd = file_rdd_raw.map(lambda x: x.split(","))
file_rdd.collect()
```

```
[['age',
  'anaemia',
  'creatinine_phosphokinase',
  'diabetes',
  'ejection_fraction',
  'high_blood_pressure',
  'platelets',
  'serum_creatinine',
  'serum_sodium',
  'sex',
  'smoking',
  'time',
  'DEATH_EVENT'],
 ['75',
  '0',
  '582',
  '0',
  '20',
  '1',
  '265000',
  '1.9',
  '130',
  '1',
  '0',
  '4',
  '1'],
 ['55',
  '0',
```

- The second data manipulation that I have used is the **filter transformation**. I removed the patients who are not deceased using the lambda function and saved it to the file_rdd_filtered variable. Using the **take** action, I was able to display 10 rows of data from the file_rdd_filtered.

```
[28]
#Filter and Take
#Removes patients who are not deceased during the Follow-up Period
file_rdd_filtered = file_rdd.filter(lambda x: x[12] != '0')
file_rdd_filtered.take(10)
```

```
 ['65',
  '1',
  '160',
  '1',
  '20',
  '0',
  '327000',
  '2.7',
  '116',
  '0',
  '0',
  '8',
  '1'],
 ['90', '1', '47', '0', '40', '1', '204000', '2.1', '132', '1', '1', '8', '1'],
 ['75',
  '1',
  '246',
  '0',
  '15',
  '0',
  '127000',
  '1.2',
  '137',
  '1',
  '0',
  '10',
  '1'],
 ['60',
```

- The third data manipulation that I have used is also the **filter transformation**. I filtered the sex column using the lambda function and saved it to the file_rdd_females and file_rdd_males variables respectively. Using the **count** and print action, I was able to display the number female and male patients from the dataset.

```
[30]
#Filter and Count
#Counts the number of males and females patients
file_rdd_females = file_rdd.filter(lambda x: x[9] == '0')
file_rdd_males = file_rdd.filter(lambda x: x[9] == '1')
print("The total number of Female patients is", file_rdd_females.count())
print("The total number of Male patients is", file_rdd_males.count())

The total number of Female patients is 105
The total number of Male patients is 194
```

- The final data manipulation that I have used is the **sortBy function**. I was able to sort the dataset using the lambda function in an ascending manner according to their age (youngest – oldest) and saved it to the file_rdd_sorted variable. Using the **collect** action, I was able to display the file_rdd_sorted by age in an ascending manner.

```
[31]
#SortBy
#Sorts the file according to the age of the patients in ascending manner
file_rdd_sorted = file_rdd.sortBy(lambda x: x[0], ascending=True)
file_rdd_sorted.collect()

[['40',
  '0',
  '478',
  '1',
  '30',
  '0',
  '303000',
  '0.9',
  '136',
  '1',
  '0',
  '148',
  '0'],
 ['40',
  '0',
  '244',
  '0',
  '45',
  '1',
  '275000',
  '0.9',
  '140',
  '0',
  '0',
  '174',
  '0'],
 ['40',
```

Reflection about data manipulation using an RDD:

At first, I had a hard time manipulating the RDD and performing different transformations and functions to it, even loading it became troublesome since I am not much familiar with Pyspark and its environment. But once I get the hang of it I encountered fewer errors. The map transformation helped a lot in completing this task due to that I cannot access specific values from my dataset since each line/row is returning as one whole string. Using the map transformation and lambda function I was able to split each row and split it by columns using a comma. Since then, it was fun manipulating RDDs and answering each question I have in mind about my dataset.

IV. Data Manipulation using a Dataframe

- Load my chosen csv file as a dataframe and save it to the variable file_df.

```python
#Load dataset as a Dataframe
file_df = spark.read.csv("heart_failure_clinical_records_dataset.csv", header=True, inferSchema=True)
```
[32]

- I started with the **select function** in which I get the basic statistics (using the **describe** function) of the chosen data columns [age, creatinine phosphokinase, ejection fraction, platelets, serum creatinine, and serum sodium] and displayed the output using the **show** function.

```python
#Select
file_df.select(['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine',
'serum_sodium']).describe().show()
```
[33]

```
+-------+-----------------+------------------------+-----------------+-----------------+-----------------+---------
---------+
|summary|              age|creatinine_phosphokinase| ejection_fraction|        platelets|  serum_creatinine|
serum_sodium|
+-------+-----------------+------------------------+-----------------+-----------------+-----------------+---------
---------+
|  count|              299|                     299|              299|              299|              299|
299|
|   mean| 60.83389297658862|       581.8394648829432| 38.08361204013378|263358.02926421416|
1.393879598662207|136.62541806020067|
| stddev|11.894809074044469|       970.2878807124358|11.834840741039168|  97804.2368685983|1.0345100640898544|
4.412477283909232|
|    min|             40.0|                      23|               14|          25100.0|              0.5|
113|
|    max|             95.0|                    7861|               80|         850000.0|              9.4|
148|
+-------+-----------------+------------------------+-----------------+-----------------+-----------------+---------
---------+
```

- Continued with the **filter function** in which I filtered the dataframe using the conditions that the age of the patient must be <= 50 and they must be already deceased, then displayed the output using the **show** function.

```python
#Filter
file_df.filter((file_df['age'] <= 50) & (file_df['DEATH_EVENT'] == '1')).show()
```
[34]

```
+----+-------+------------------------+--------+-----------------+------------------+---------+----------------+---
---------+---+-------+----+-----------+
|
age|anaemia|creatinine_phosphokinase|diabetes|ejection_fraction|high_blood_pressure|platelets|serum_creatinine|serum_so
+----+-------+------------------------+--------+-----------------+------------------+---------+----------------+---
---------+---+-------+----+-----------+
|50.0|      1|                     111|       0|               20|                 0| 210000.0|             1.9|
137|  1|      0|   7|          1|
|45.0|      1|                     981|       0|               30|                 0| 136000.0|             1.1|
137|  1|      0|  11|          1|
|50.0|      1|                     168|       0|               38|                 1| 276000.0|             1.1|
137|  1|      0|  11|          1|
|45.0|      0|                     582|       0|               14|                 0| 166000.0|             0.8|
127|  1|      0|  14|          1|
|48.0|      1|                     582|       1|               55|                 0|  87000.0|             1.9|
121|  0|      0|  15|          1|
|50.0|      1|                     249|       1|               35|                 1| 319000.0|             1.0|
128|  0|      0|  28|          1|
|50.0|      0|                     124|       1|               30|                 1| 153000.0|             1.2|
136|  0|      1|  32|          1|
|50.0|      0|                     582|       1|               38|                 0| 310000.0|             1.9|
135|  1|      1|  35|          1|
|49.0|      0|                     789|       0|               20|                 1| 319000.0|             1.1|
```

- Followed with the **groupBy function** in which I grouped the patients by age and get the corresponding means of their [serum creatinine, serum sodium, and platelets], displayed the output using the **show** function.

```
[35]
#GroupBy
file_df_by_age = file_df.groupby('age').mean().select(['age', 'avg(serum_creatinine)', 'avg(serum_sodium)',
'avg(platelets)']).show()
```

```
+----+--------------------+-----------------+-----------------+
| age|avg(serum_creatinine)| avg(serum_sodium)|   avg(platelets)|
+----+--------------------+-----------------+-----------------+
|70.0|              1.2532|           137.32|      258858.3212|
|67.0|                1.19|            135.0|       239179.015|
|69.0|  1.9000000000000001|134.33333333333334|199666.66666666666|
|49.0|  0.9750000000000001|            136.0|         286500.0|
|75.0|  1.7018181818181817|134.36363636363637|254097.64454545453|
|64.0|  1.6333333333333335|135.66666666666666| 265666.6666666667|
|47.0|                 0.8|            134.0|         130000.0|
|42.0|  1.4685714285714284|137.14285714285714|244051.14714285714|
|44.0|                1.15|            134.5|       249179.015|
|62.0|  0.9199999999999999|            138.2|         231200.0|
|80.0|  3.0428571428571436|136.71428571428572|233051.14714285714|
|86.0|                1.83|            134.0|        263358.03|
|94.0|                1.83|            134.0|        263358.03|
|41.0|                 0.8|            140.0|         374000.0|
|85.0|  1.7666666666666668|            134.0| 306166.6666666667|
|77.0|  1.4500000000000002|            141.0|         314500.0|
|50.0|  1.0733333333333335|136.14814814814815| 257939.1862962963|
|56.0|                 1.7|            140.0|         133000.0|
|78.0|  1.0499999999999998|            137.5|         379000.0|
|79.0|                 1.8|            133.0|         172000.0|
+----+--------------------+-----------------+-----------------+
```

- Finally, with the **orderBy function** in which I just simply organized the dataframe in a descending manner according to the patient's age (oldest – youngest) and displayed the output using the **show** function.

```
[53]
#OrderBy
from pyspark.sql.functions import desc
file_df.orderBy(desc('age')).show()
```

```
+----+-------+------------------------+--------+-----------------+------------------+---------+----------------+---
---------+---+-------+----+-----------+
|
age|anaemia|creatinine_phosphokinase|diabetes|ejection_fraction|high_blood_pressure|platelets|serum_creatinine|serum_so
+----+-------+------------------------+--------+-----------------+------------------+---------+----------------+---
---------+---+-------+----+-----------+
|95.0|      1|                     112|       0|               40|                 1| 196000.0|             1.0|
138|  0|      0|  24|          1|
|95.0|      1|                     371|       0|               30|                 0| 461000.0|             2.0|
132|  1|      0|  50|          1|
|94.0|      0|                     582|       1|               38|                 1|263358.03|            1.83|
134|  1|      0|  27|          1|
|90.0|      1|                      47|       0|               40|                 1| 204000.0|             2.1|
132|  1|      1|   8|          1|
|90.0|      1|                     337|       0|               38|                 0| 390000.0|             0.9|
144|  0|      0| 256|          0|
|90.0|      1|                      60|       1|               50|                 0| 226000.0|             1.0|
134|  1|      0|  30|          1|
|87.0|      1|                     149|       0|               38|                 0| 262000.0|             0.9|
140|  1|      0|  14|          1|
|86.0|      0|                     582|       0|               38|                 0|263358.03|            1.83|
134|  0|      0|  95|          1|
|85.0|      0|                      23|       0|               45|                 0| 360000.0|             3.0|
```

Reflection about manipulating data from a dataframe using SQL:

Compared to data manipulation using an RDD, I guess I can safely say that I find manipulating data from a dataframe using SQL a lot easier. I did not encounter as many problems as I had with RDD. It is also a lot more straightforward on the side of the dataframe since you can even code one liners and get the exact same result from a lengthy line of code using an RDD. I also enjoy how the outputs of dataframes are more structured and cleaner to the user's eyes. Since we're working with data, visual input is our primary sense for conceiving ideas.

My Three Q's:

```
[76]
#Question 1: What is the mean of smokers and death events inline with the patient's age?
file_df.groupby('age').mean().select(['age','avg(smoking)','avg(DEATH_EVENT)']).show()
```

```
+----+-------------------+-------------------+
| age|       avg(smoking)|   avg(DEATH_EVENT)|
+----+-------------------+-------------------+
|70.0|                0.4|               0.28|
|67.0|                0.5|                0.0|
|69.0| 0.6666666666666666| 0.6666666666666666|
|49.0|               0.25|               0.25|
|75.0|0.18181818181818182| 0.5454545454545454|
|64.0|                0.0|                0.0|
|47.0|                0.0|                0.0|
|42.0|0.42857142857142855|0.14285714285714285|
|44.0|                0.5|                0.0|
|62.0|                0.4|                0.2|
|80.0|0.42857142857142855| 0.7142857142857143|
|86.0|                0.0|                1.0|
|94.0|                0.0|                1.0|
|41.0|                1.0|                0.0|
|85.0| 0.3333333333333333|                0.5|
|77.0|                0.0|                0.5|
|50.0| 0.3333333333333333| 0.2962962962962963|
|56.0|                0.0|                0.0|
|78.0|                1.0|                0.0|
|79.0|                0.0|                0.0|
+----+-------------------+-------------------+
```

- Question 1 has a straightforward question and answer that can be seen at the table.

```
[79]
#Question 2: How many deaths occurred for both sexes?
file_df.selectExpr('DEATH_EVENT == 1 as deceased', 'DEATH_EVENT','sex').groupBy('sex').pivot('deceased',
('true', 'false')).count().show()
```

```
+---+----+-----+
|sex|true|false|
+---+----+-----+
|  1|  62|  132|
|  0|  34|   71|
+---+----+-----+
```

- As for Question 2, the deaths that occurred for the females were 34, and as for the males, they recorded a death of 64. If we combine all the deaths and non-deaths of both sexes, we will arrive at answer of 299, which is the exact number of patients in this dataset.

```
[81]
#Question 3: Is there a significant difference between the actual death events values and the predictions yielded from implementing a Linear Regression Model to the same dataset?
from pyspark.ml.feature import VectorAssembler
featureAssembler = VectorAssembler(inputCols=
['age','anaemia','creatinine_phosphokinase','diabetes','ejection_fraction','high_blood_pressure',
'platelets','serum_creatinine','serum_sodium','sex','smoking','time'],outputCol='Independent Features')

output = featureAssembler.transform(file_df)
output.select('Independent Features').show(10)
```

```
+--------------------+
|Independent Features|
+--------------------+
|[75.0,0.0,582.0,0...|
|[55.0,0.0,7861.0,...|
|[65.0,0.0,146.0,0...|
|[50.0,1.0,111.0,0...|
|[65.0,1.0,160.0,1...|
|[90.0,1.0,47.0,0....|
|[75.0,1.0,246.0,0...|
|[60.0,1.0,315.0,1...|
|[65.0,0.0,157.0,0...|
|[80.0,1.0,123.0,0...|
+--------------------+
only showing top 10 rows
```

```
[82]
finalized_data = output.select('Independent Features','DEATH_EVENT')
finalized_data.show(30)
```

```
+--------------------+-----------+
|Independent Features|DEATH_EVENT|
+--------------------+-----------+
|[75.0,0.0,582.0,0...|          1|
|[55.0,0.0,7861.0,...|          1|
|[65.0,0.0,146.0,0...|          1|
|[50.0,1.0,111.0,0...|          1|
|[65.0,1.0,160.0,1...|          1|
|[90.0,1.0,47.0,0....|          1|
|[75.0,1.0,246.0,0...|          1|
|[60.0,1.0,315.0,1...|          1|
|[65.0,0.0,157.0,0...|          1|
|[80.0,1.0,123.0,0...|          1|
|[75.0,1.0,81.0,0....|          1|
|[62.0,0.0,231.0,0...|          1|
|[45.0,1.0,981.0,0...|          1|
|[50.0,1.0,168.0,0...|          1|
|[49.0,1.0,80.0,0....|          0|
|[82.0,1.0,379.0,0...|          1|
|[87.0,1.0,149.0,0...|          1|
|[45.0,0.0,582.0,0...|          1|
|[70.0,1.0,125.0,0...|          1|
|[48.0,1.0,582.0,1...|          1|
|[65.0,1.0,52.0,0....|          0|
```

```
from pyspark.ml.regression import LinearRegression
train_data, test_data = finalized_data.randomSplit([0.75,0.25])
regressor = LinearRegression(featuresCol='Independent Features', labelCol='DEATH_EVENT')
regressor = regressor.fit(train_data)

pred_results=regressor.evaluate(test_data)
pred_results.predictions.show(30)
```
[83]

```
+--------------------+-----------+--------------------+
|Independent Features|DEATH_EVENT|          prediction|
+--------------------+-----------+--------------------+
|[42.0,0.0,582.0,0...|          0|  0.1456599637517586|
|[42.0,0.0,5209.0,...|          0|   0.453549400875807|
|[42.0,1.0,250.0,1...|          1|  0.6306310855972344|
|[45.0,0.0,582.0,0...|          1|  0.7644950325451949|
|[45.0,0.0,582.0,1...|          0| -0.0839943674404251|
|[45.0,0.0,2413.0,...|          0|  -0.126302475543266|
|[46.0,0.0,168.0,1...|          1|  0.6906856531390787|
|[49.0,0.0,789.0,0...|          1|  0.5372446809871239|
|[50.0,0.0,250.0,0...|          0|  0.3411650325103355|
|[50.0,1.0,115.0,0...|          0| 0.28863976384935297|
|[50.0,1.0,121.0,1...|          0|  0.1300754522547709|
|[50.0,1.0,168.0,0...|          1| 0.48907542031856166|
|[50.0,1.0,582.0,1...|          0| 0.28236929895477036|
|[51.0,1.0,582.0,1...|          0| 0.29074210981787996|
|[52.0,0.0,132.0,0...|          0| 0.29884572713963853|
|[52.0,1.0,58.0,0....|          0| 0.33783649056937026|
|[53.0,0.0,63.0,1....|          0| 0.32234770808969526|
|[53.0,1.0,707.0,0...|          0|0.057614038994489025|
|[55.0,0.0,60.0,0....|          0|  0.3858643721045092|
|[55.0,0.0,66.0,0....|          0|-0.05376279413756846|
|[55.0,0.0,109.0,0...|          0|  0.4186060439222479|
|[55.0,0.0,7861.0,...|          1|  0.7964612056960994|
|[55.0,1.0,180.0,0...|          0|-0.0294108733283 7...|
|[57.0,1.0,115.0,0...|          0| 0.41837735239518836|
|[57.0,1.0,129.0,0...|          1|   0.526534282360763|
```

- In Question 3, there is not much of a significant difference between the actual death event column data and the linear regression model prediction that was created in terms of yielded values. I would say that even though you need to round off the predictions to get 1, I think it is still a somewhat good prediction model even though there were some lapses or inaccuracies with it, which we cannot change from time to time.

V. Synthesis and Moving Forward

A. What are the things that you learned about Big Data?

First and foremost, I have learned the value of data, especially big data whether it is structured or unstructured in solving problems that we face in our daily lives. We can now go beyond and predict unimaginable things that seems impossible to us from the past decades. I think for the coming years big data will soon be much more recognized in the field. It is amazing how big data can answer almost every question we have, we just have to transform it to something that all of us can understand.

B. How do you plan to use the things you learned moving forward?

This undertaking helped me fell in love more with my chosen program. It feels like this is the closest thing that I have experienced that is somewhat similar to what actual professionals do in the field. Moving forward, I will continue to broaden my skillset and improve the skills & knowledge I already have by practicing & taking on small projects.

C. What are your areas for improvement?

I guess I still have to improve my coding skills. Experiencing that RDD helped me realize that I still have a long way to go in programming. Apart from that, I think there is nothing studying, practicing, and a little perseverance cannot solve.