Introduction

In this project, I will be explaining how to set up and use **PYSPARK** in a machine learning instance. All reports and analysis will be discussed real time.

All visualizations of analysis will be done with Tableau. The dataset used in this project is on Accident Anlaysis in the UK from 2005-2014.

The structure of this paper which is in line with the goals of the projects is as follows:

In this paper I am going to explain and demonstrate how to set up and use PySpark for prediction and building a pipeline. Next, the dataset used for the demonstration is introduced. It is data about accidents in the UK and it is a public dataset. The demonstration part of this paper is structured as follows. First, the steps and goals of the task and the reasons for them are explained. Second, the crucial parts of the code are showed and explained. Only small portion of the code is showed.

The complete source code is included in the appendix A. The code is attached to the document through a github link and also appended to this document. First, the process of cleaning the data is described. Next, we perform a detailed explorative data analysis and produce some visualisations of the findings. Finally, we will train a models using Pyspark for accident severity prediction and also resting its accuracy.

**DATASET ANALYSIS AND GOALS**

To demonstrate how PYSPARK can be used for analysis alongside Tableau for visualizations while performing even the most complex of tasks doing them. The dataset chosen was recorded in the UK, the stats set covers basic details about the accident and also likely factors that could have contributed to the accident.

It can be accessed publicly on [Kaggle](https://www.kaggle.com/datasets/devansodariya/road-accident-united-kingdom-uk-dataset). It covers accidents from 2005-2012. It has 1,048,576 observations and 33 features.

The table below lists all the features of the dataset and a brief explanation about the type.

|  |  |
| --- | --- |
| FEATURE | DATA TYPE |
| unknown | Numeric |
| Accident Index | Numeric |
| Location\_Easting\_OSGR | Numeric |
| Location\_Northing\_OSGR | Numeric |
| Longitude | Numeric |
| Latitude | Numeric |
| Police\_Force | Numeric |
| Accident\_Severity | Numeric |
| Number of Vehicles | Numeric |
| Number\_of\_Casualties | Numeric |
| Date | String |
| Day\_of\_Week | Numeric |
| Time | Numeric |
| Local\_Authority (District) | Numeric |
| Local\_Authority (Highway) | Numeric |
| 1st Road Class | String |
| Road\_Type | String |
| Speed\_limit | Numeric |
| Junction\_Control | Numeric |
| 2nd\_Road\_class |  |
| 2nd\_Road\_Number |  |
| Pedestrian Crossing Human Control |  |
| Pedestrian Crossing-Physical Facilities |  |
| Light Conditions |  |
| Weather Conditions |  |

There is a mix of Numeric (integers, float, doubles), String or Text, Categorical as shown above. Also, since most of the information were entered manually. A good number of this information will be missing.

The analysis of this dataset is to tell the story and point to indicating factors and explore the data fully while visualizing it with tableau and also look at the most indicating factors of the accidents recorded While taking cognizance of the casualties and severity.

The second part of this analysis of the dataset is to analyze and predict accident severity while creating a model with Pyspark exclusively.

At the end of the analysis, we will also try to see if the factors can be worked on to reduce accidents and the loss of lives and properties attached to it. It can also help in knowing where to deploy more assistance, if possible, medical aids and traffic wardens.

1. import numpy as np # linear algebra
2. import matplotlib.pyplot as plt
3. import seaborn as sns
4. import sklearn
5. import random
6. import os
8. from pyspark.sql import SparkSession
9. from pyspark.ml  import Pipeline
10. from pyspark.sql import SQLContext
11. from pyspark.sql.functions import mean,col,split, col, regexp\_extract, when, lit, isnan, count
12. from pyspark.ml.feature import StringIndexer, VectorAssembler
13. from pyspark.ml.evaluation import MulticlassClassificationEvaluator
14. from pyspark.ml.feature import QuantileDiscretizer

Importing the needed frameworks and modules.

**DATA PREPROCESSING**

The data was manually inputted and so we expect some part of the data missing, in order not to affect our analysis with this, we will be dropping rows with any missing data but not after we fully check the rows missing data. Some of the rows can be replaced and we will replace them accordingly only the ones that can’t be filled will be dropped.

In Line 1, we can see that we started the PYSPARK session with the syntax.

Line 3, shows us how to import the data the same way we do it in pandas.

1. spark = SparkSession.builder.appName('accident\_prediction').getOrCreate()
2. # After creating spark, we use spark.read.csv to read dataset, like pandas.read\_csv
3. df = spark.read.csv('./UK\_Accident.csv',header = 'True',inferSchema='True')

This allows us to import our data, but we can specify our schema, in the code above we are not specifying our schema. So we can show our imported schema with a line of code.

1. df.printSchema()

We have this as our output.

1. root
2. |-- \_c0: integer (nullable = true)
3. |-- Accident\_Index: string (nullable = true)
4. |-- Location\_Easting\_OSGR: double (nullable = true)
5. |-- Location\_Northing\_OSGR: double (nullable = true)
6. |-- Longitude: double (nullable = true)
7. |-- Latitude: double (nullable = true)
8. |-- Police\_Force: integer (nullable = true)
9. |-- Accident\_Severity: integer (nullable = true)
10. |-- Number\_of\_Vehicles: integer (nullable = true)
11. |-- Number\_of\_Casualties: integer (nullable = true)
12. |-- Date: string (nullable = true)
13. |-- Day\_of\_Week: integer (nullable = true)
14. |-- Time: string (nullable = true)
15. |-- Local\_Authority\_(District): integer (nullable = true)
16. |-- Local\_Authority\_(Highway): string (nullable = true)
17. |-- 1st\_Road\_Class: integer (nullable = true)
18. |-- 1st\_Road\_Number: integer (nullable = true)
19. |-- Road\_Type: string (nullable = true)
20. |-- Speed\_limit: integer (nullable = true)
21. |-- Junction\_Control: string (nullable = true)
22. |-- 2nd\_Road\_Class: integer (nullable = true)
23. |-- 2nd\_Road\_Number: integer (nullable = true)
24. |-- Pedestrian\_Crossing-Human\_Control: string (nullable = true)
25. |-- Pedestrian\_Crossing-Physical\_Facilities: string (nullable = true)
26. |-- Light\_Conditions: string (nullable = true)
27. |-- Weather\_Conditions: string (nullable = true)
28. |-- Road\_Surface\_Conditions: string (nullable = true)
29. |-- Special\_Conditions\_at\_Site: string (nullable = true)
30. |-- Carriageway\_Hazards: string (nullable = true)
31. |-- Urban\_or\_Rural\_Area: integer (nullable = true)
32. |-- Did\_Police\_Officer\_Attend\_Scene\_of\_Accident: string (nullable = true)
33. |-- LSOA\_of\_Accident\_Location: string (nullable = true)
34. |-- Year: integer (nullable = true)

As seen as above this is our schema, which shows us the data types we have in our dataset.

Let’s check for our missing rows and try to fill it in.

1. df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]
2. ).show(truncate=False)

The output will be added to the source code attached to this analysis.

1. fin\_df = df
3. fin\_df.na.fill(value='Unknown',subset=["Weather\_Conditions"])
4. fin\_df.na.fill(value='Normal',subset=["Road\_Surface\_Conditions"])
5. fin\_df.na.fill(value= 0 ,subset=["Latitude"])
6. fin\_df.na.fill(value='None',subset=["Special\_Conditions\_at\_Site"])
7. fin\_df.na.fill(value='None',subset=["Carriageway\_Hazards"])
8. fin\_df.na.fill(value='No',subset=["Did\_Police\_Officer\_Attend\_Scene\_of\_Accident"])
9. fin\_df.na.fill(value='None',subset=["Junction\_Control"])
10. fin\_df.na.fill(value='0',subset=["Location\_Easting\_OSGR"])

We can still check through and see those not dropped as shown in the dialog box before now. Then we can use the syntax to drop the rows that can’t be replaced. PYSPARK has a direct way to dropping rows and this is demonstrated in the code below

1. fin\_df = fin\_df.dropna()

**EXPLORATION OF DATA /EXPLORATIVE ANALYSIS**

In this phase we want to learn everything we can learn about the data and visualize with Tableau.

In this section, we will be using Tableau exclusively to visualize. Tableau is a visualizing software, with little or no code that helps us to tell and visualize a dataset faster and more exquisitely than using native coding languages. Most of the computation was done with PYSPARK, we will only try to do the visualizations of this data in Tableau as indicated above.

**VISUALIZATIONS OF DATA AND EXPLANATIONS OF INDICATING FACTORS**.

Since this dataset has years recorded, we will need to find the years with the highest number of casualties and accident severity, excluding all factors. The code to manipulate the data and get the needed columns is added below.

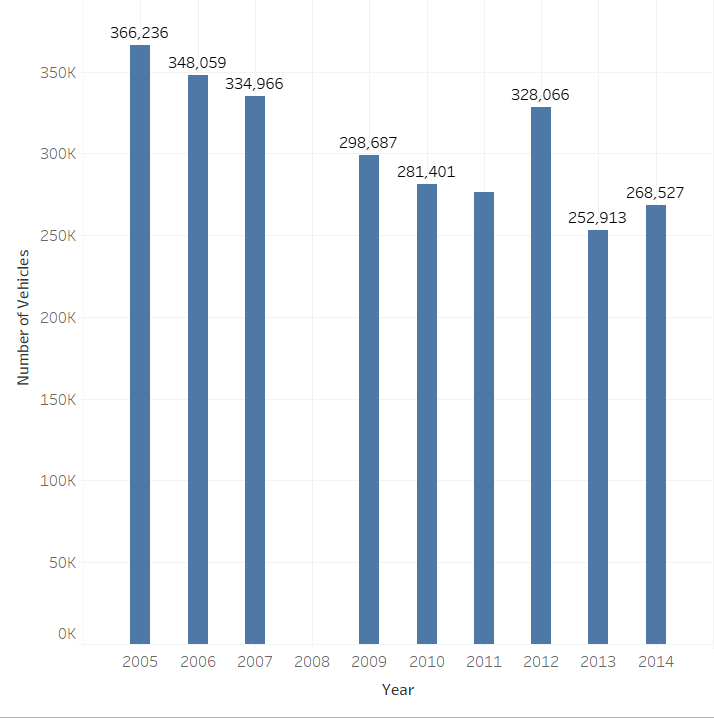
Let’s take a look at the distinct years we have and all distinct values we have.

1. fin\_df.select('Year').distinct().collect()
3. fin\_df.groupBy("Year").sum("Number\_of\_Casualties").show(truncate=False)
4. fin\_df.groupBy("Year").sum("Accident\_Severity").show(truncate=False)
5. fin\_df.groupBy("Year").sum("Number\_of\_Vehicles").show(truncate=False)

We’ve formatted the dataframe, let’s visualize it with Tableau

As visualized on the data frame, we can see that the year 2005 has the highest severity with the severity as high as 563,350

Next up is 508,587 which is in year 2012



As seen in the data visualized above, we can see that the years are in tally with the numbers of vehicles, the casualties recorded and the severity on the same scale and magnitude.

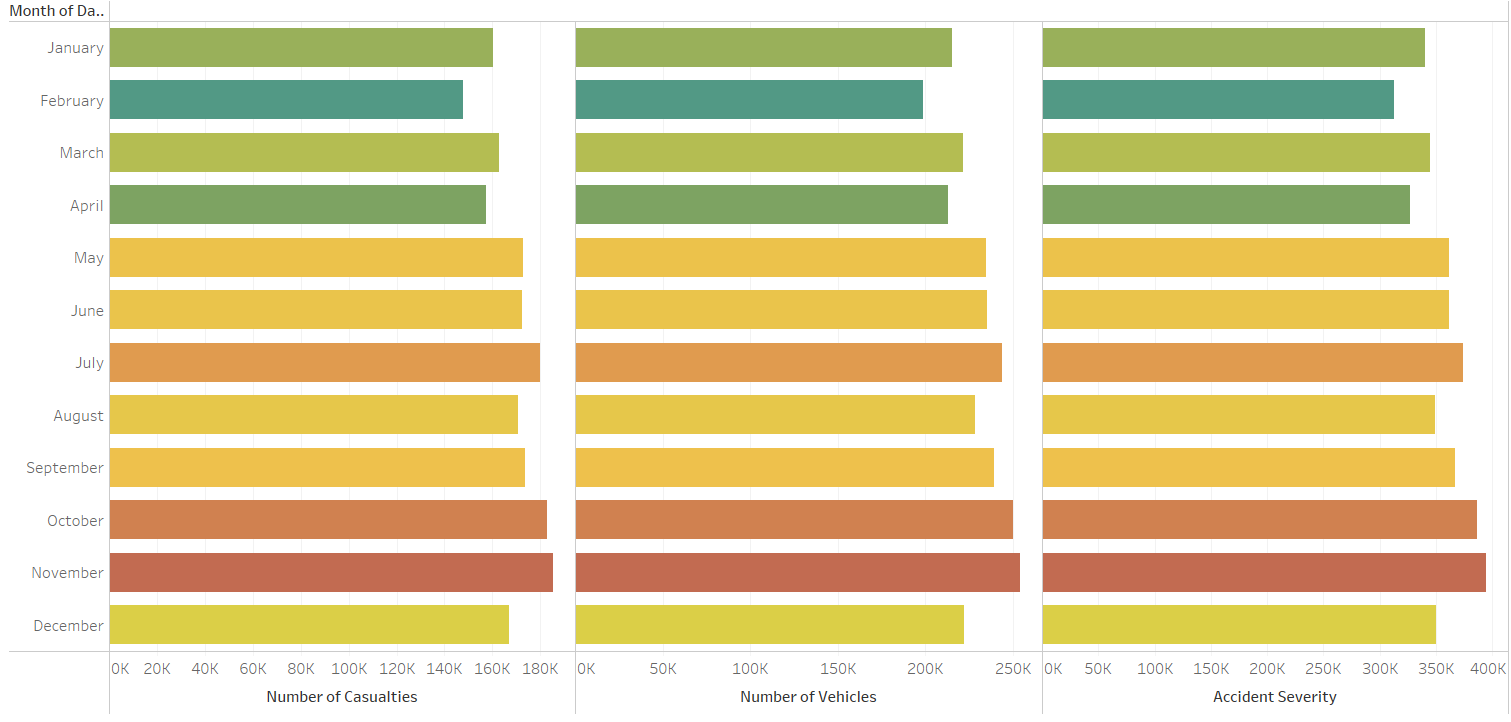
As we can see the accidents over the years have reduced but we still have our peak year in 2005 which is not surprisingly the first year in this chart. We will notice year 2008 is missing.

Let’s do a simple breakdown of this, we will continue to visualize with tableau taking note of the other categorical factors.

We are breaking the data down with respect to time,

We’ve started with the year, we will go on to quarters, months, Day of the week, Hour of the day.

All this is possible with tableau and allows more flexibility and ease than using the native plot functions of pyspark.

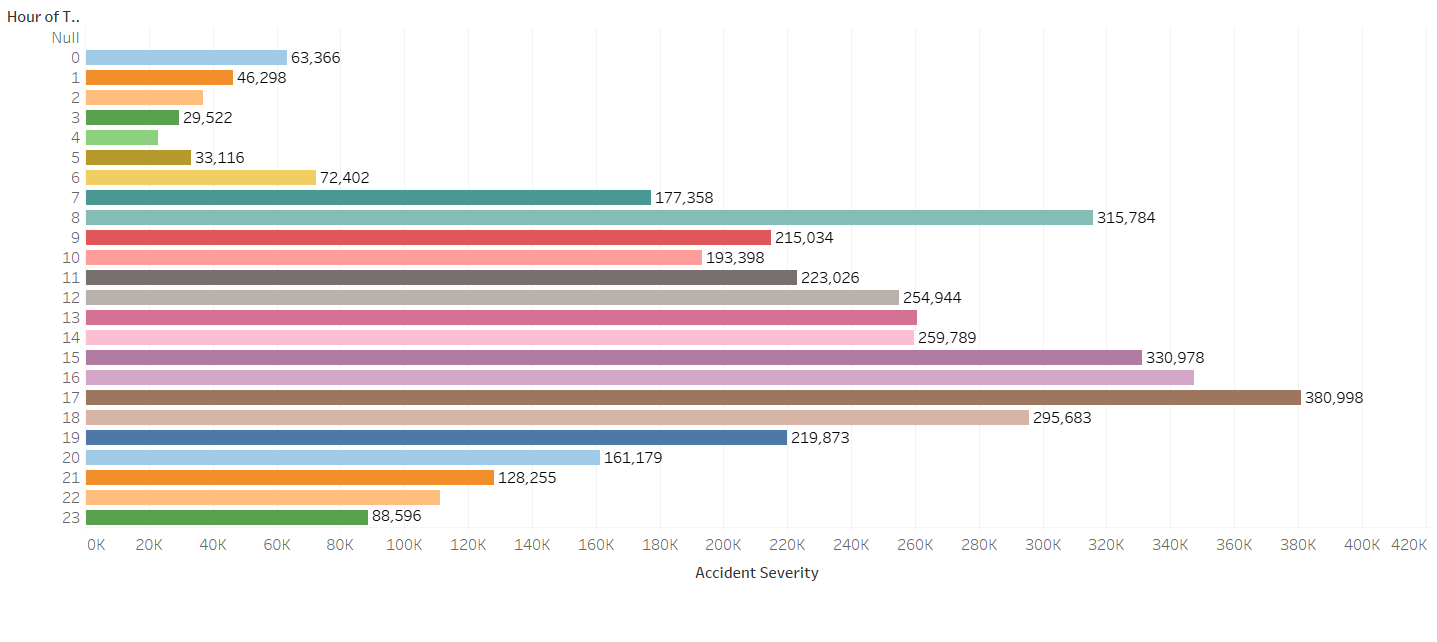


now that we have fully visualized this we can pinpoint to what time and period we have more casualties and severity

As seen above, we can fully pinpoint our top 3 months in terms of accident casualties, accident severity and number of vehicles, which are November, October, July respectively. These months are actually the top months. Let’s check what day of the week and also time of the day we have the most likelihood for the accidents to occur.

INDICATING FACTORS

Now that we have the timing, let’s go to the indicating factors of this. Since we have our anlalysis according to months, let’s see where and what causes these accidents to happen. Let’s also check what time of the day and week accidents are likely to occur at this particular point.



As seen above, we can see that our peak period/ time is during the estimated rush period when coming back from work which is at 5pm (17;00 hrs) and the next is at 3pm (15:00 hrs) followed by 8am in the morning. We can attribute this to the rush hour period when we have quite a number of vehicles on the road. If the year and time is anything to go by, we will agree that we will have the same number of vehicles on the road at this particular point in time and also the same casualty level.

A full dashboard pointing to the factors is attached below.

MODEL TRAINING

1. #Accident Severity
2. sev\_acc = fin\_df.select(col('Accident\_Index'), col("Accident\_Severity"),col("Day\_of\_Week"),col("Road\_Surface\_Conditions"),col("Light\_Conditions"), col("Weather\_Conditions"), col('Number\_of\_Casualties'), col("Date"), col("Road\_Type"), col('LSOA\_of\_Accident\_Location'))
3. acc\_sev = fin\_df.select(col("Accident\_Severity"),col("Day\_of\_Week"),col("Road\_Surface\_Conditions"),col("Light\_Conditions"), col("Weather\_Conditions"), col('Number\_of\_Casualties'))

In our model training we selected the part of the code that is needed for us and also we can easily work with due to its relevance to the accident scene. Some of the columns were ignored because they were right after the accident occurred or just normal indications which was after the accidents occurred and had little or no direct effect to the accident.

So we worked on getting the correlation matrix. The remaining part of the code is inserted afterwards. With the output.

1. bindexer = StringIndexer(inputCols=["Road\_Surface\_Conditions", "Light\_Conditions", "Weather\_Conditions"],
2. outputCols=["Road\_Conditions\_indexed","Light\_Conditions\_indexed","Weather\_Conditions\_indexed"])
3. bindexed = bindexer.fit(acc\_sev).transform(acc\_sev).drop( "Road\_Surface\_Conditions", "Light\_Conditions", "Weather\_Conditions")
4. bindexed.limit(5).toPandas()
6. # acc\_sev.sample()
7. bindexed.show()
9. +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+ |Accident\_Severity|Day\_of\_Week|Number\_of\_Casualties|Road\_Conditions\_indexed|Light\_Conditions\_indexed|Weather\_Conditions\_indexed| +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+ | 2| 3| 1| 1.0| 0.0| 1.0| | 3| 4| 1| 0.0| 1.0| 0.0| | 3| 5| 1| 0.0| 1.0| 0.0| | 3| 6| 1| 0.0| 0.0| 0.0| | 3| 2| 1| 1.0| 3.0| 0.0| | 3| 3| 1| 1.0| 0.0| 1.0| | 3| 5| 1| 0.0| 1.0| 0.0| | 3| 6| 2| 0.0| 0.0| 0.0| | 3| 7| 2| 0.0| 1.0| 0.0| | 3| 7| 5| 0.0| 0.0| 0.0| | 3| 1| 1| 0.0| 1.0| 0.0| | 3| 3| 1| 1.0| 1.0| 0.0| | 3| 3| 1| 1.0| 0.0| 1.0| | 3| 3| 1| 1.0| 1.0| 1.0| | 3| 3| 2| 0.0| 0.0| 0.0| | 3| 3| 1| 0.0| 0.0| 0.0| | 2| 5| 1| 0.0| 1.0| 0.0| | 3| 6| 1| 0.0| 0.0| 0.0| | 3| 6| 1| 0.0| 1.0| 0.0| | 2| 7| 1| 0.0| 1.0| 0.0| +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+ only showing top 20 rows

We converted the string columns to numeric with string indexer in pyspark as seen above

1. from pyspark.ml.stat import Correlation
2. from pyspark.ml.feature import VectorAssembler
4. # convert to vector column first
5. vector\_col = "features"
6. assembler = VectorAssembler(inputCols= ["Day\_of\_Week", 'Number\_of\_Casualties', 'Road\_Conditions\_indexed',
7. 'Light\_Conditions\_indexed', 'Weather\_Conditions\_indexed'], outputCol='features')
8. df\_vector = assembler.transform(bindexed).select(vector\_col,'Accident\_Severity')        ##\*\*\*Check this out
10. matrix = Correlation.corr(df\_vector, vector\_col)
12. matrix.collect()[0]["pearson({})".format(vector\_col)].values

We vectorized the columns to find the features afterwards

1. rray([ 1. , 0.00332849, -0.07994414, 0.01355863, -0.05936759, 0.01634808, 0.00332849, 1. , -0.00177767, -0.00753256, 0.00662725, -0.00103654, -0.07994414, -0.00177767, 1. , 0.02912377, 0.03577978, 0.00727531, 0.01355863, -0.00753256, 0.02912377, 1. , 0.16799007, 0.42856646, -0.05936759, 0.00662725, 0.03577978, 0.16799007, 1. , 0.12200156, 0.01634808, -0.00103654, 0.00727531, 0.42856646, 0.12200156, 1. ])
3. df\_vector.show()

CORRELATION MATRIX

TRAINING A MODEL

1. train, test = bindexed.randomSplit([0.75,0.25])
2. from pyspark.ml.recommendation import ALS
4. #Training the accident\_prediction model using train datatset
5. rec=ALS( maxIter=10
6. ,regParam=0.01
7. ,userCol='Day\_of\_Week'
8. ,itemCol='Accident\_Severity'
9. ,ratingCol='Number\_of\_Casualties'
10. ,nonnegative=True
11. ,coldStartStrategy="drop")
13. #fit the model on train set
14. rec\_model=rec.fit(train)
16. #making predictions on test set
17. predicted\_ratings=rec\_model.transform(test)
18. predicted\_ratings.limit(5).show()
20. +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+----------+ |Accident\_Severity|Day\_of\_Week|Number\_of\_Casualties|Road\_Conditions\_indexed|Light\_Conditions\_indexed|Weather\_Conditions\_indexed|prediction| +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+----------+ | 1| 1| 1| 0.0| 0.0| 0.0| 2.0147429| | 1| 1| 1| 0.0| 0.0| 0.0| 2.0147429| | 1| 1| 1| 0.0| 0.0| 0.0| 2.0147429| | 1| 1| 1| 0.0| 0.0| 0.0| 2.0147429| | 1| 1| 1| 0.0| 0.0| 0.0| 2.0147429| +-----------------+-----------+--------------------+-----------------------+------------------------+--------------------------+----------+
22. from pyspark.ml.evaluation import RegressionEvaluator
23. # create Regressor evaluator object for measuring accuracy
24. evaluator=RegressionEvaluator(metricName='rmse',predictionCol='prediction',labelCol='Number\_of\_Casualties')
25. # apply the RE on predictions dataframe to calculate RMSE
26. rmse=evaluator.evaluate(predicted\_ratings)
27. # print RMSE error
28. print(rmse)

The rest of the code ca be found on the [github](https://github.com/esther-anyanwu)