Introduction

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

“PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark’s features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core” as found on the documentation website.

For the sake of this project, we are using Pyspark 3.0. 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 we are explaining and demonstrating how to set up and use PySpark for prediction and building a pipeline. Next, the dataset used for the demonstration is introduced. 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 code is shown.

The complete source code is included in the appendix. 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 model 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.

|  |
| --- |
| FEATURES |
| Unknown |
| Accident Index |
| Location\_Easting\_OSGR |
| Location\_Northing\_OSGR |
| Longitude |
| Latitude |
| Police\_Force |
| Accident\_Severity |
| Number of Vehicles |
| Number\_of\_Casualties |
| Date |
| Day\_of\_Week |
| Time |
| Local\_Authority (District) |
| Local\_Authority (Highway) |
| 1st Road Class |
| Road\_Type |
| Speed\_limit |
| Junction\_Control |
| 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.

I installed Pyspark with a simple line of code in my CMD terminal

1. pip install pyspark

It can be downloaded manually on the [documentation website](https://spark.apache.org/docs/latest/api/python/getting_started/install.html#manually-downloading).

We imported the extra packages needed to run the code and analysis, others will be imported as the program goes on.

1. import matplotlib.pyplot as plt
2. import seaborn as sns
3. import sklearn
4. import random
5. import os
7. from pyspark.sql import SparkSession
8. from pyspark.ml  import Pipeline
9. from pyspark.sql import SQLContext
10. from pyspark.sql.functions import mean,col,split, col, regexp\_extract, when, lit, isnan, count
11. from pyspark.ml.feature import StringIndexer, VectorAssembler
12. from pyspark.ml.evaluation import MulticlassClassificationEvaluator
13. 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 rows can be replaced, the ones that can’t will be dropped

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

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. This line of code below is used to print or display the schema of the DataFrame in the tree format along with column name and data type.

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)

The 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]).show(truncate=False)

The output is in the source code. The “.show()” function is is used to display the contents of the DataFrame in a Table Row & Column Format. By default, it shows only 20 Rows, and the column values are truncated at 20 characters. We can set our limit to be shown.

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

**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 find the years with the highest number of casualties and accident severity, excluding all factors. The code manipulates the data and gets the needed columns as shown below.

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)

Figure 1: Accident Severity per year

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

Figure 2b: Number of Vehicles per year

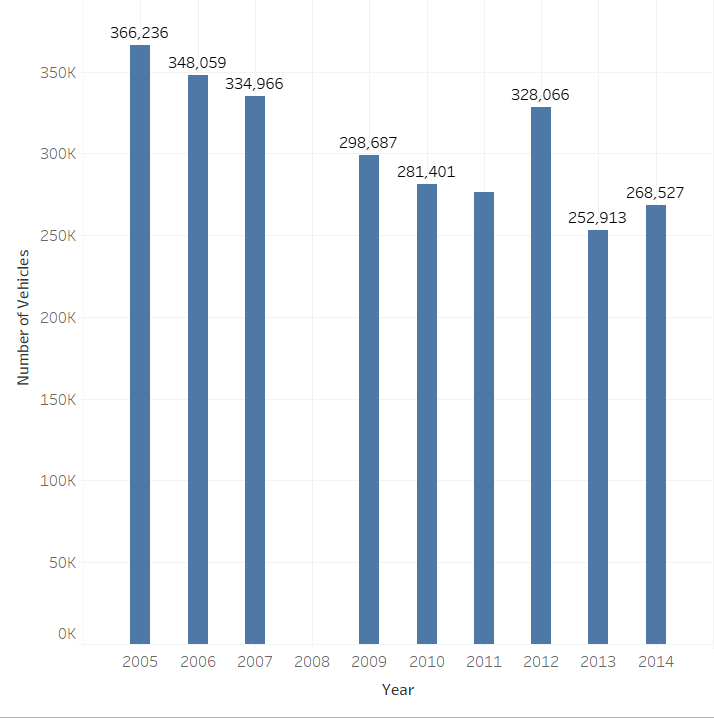


Figure 3: Number of Casualties per Year

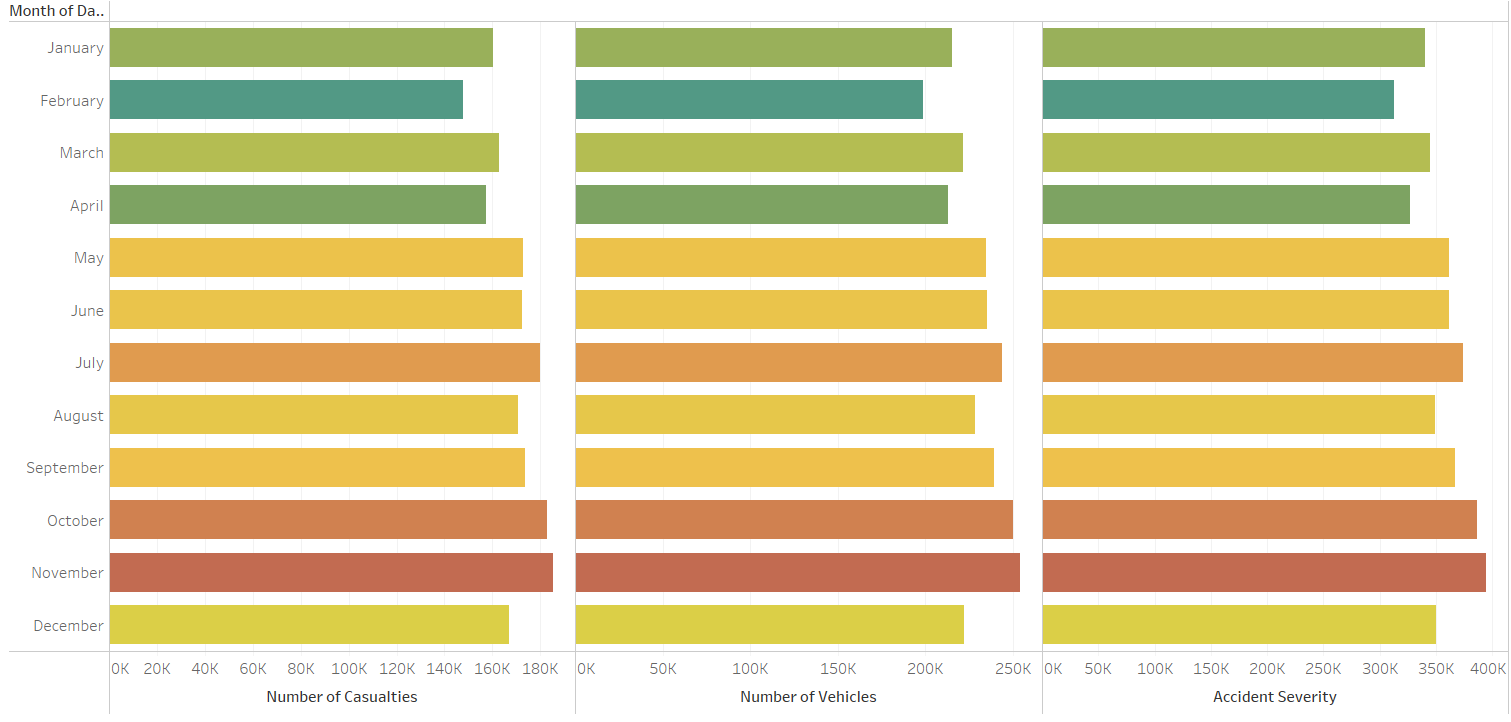
As seen in the data visualized above, we can see that the years are in tally with the numbers of vehicles, the casualties recorded and the severity on the same scale and magnitude.

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,

Figure 4Accident breakdown according to Months



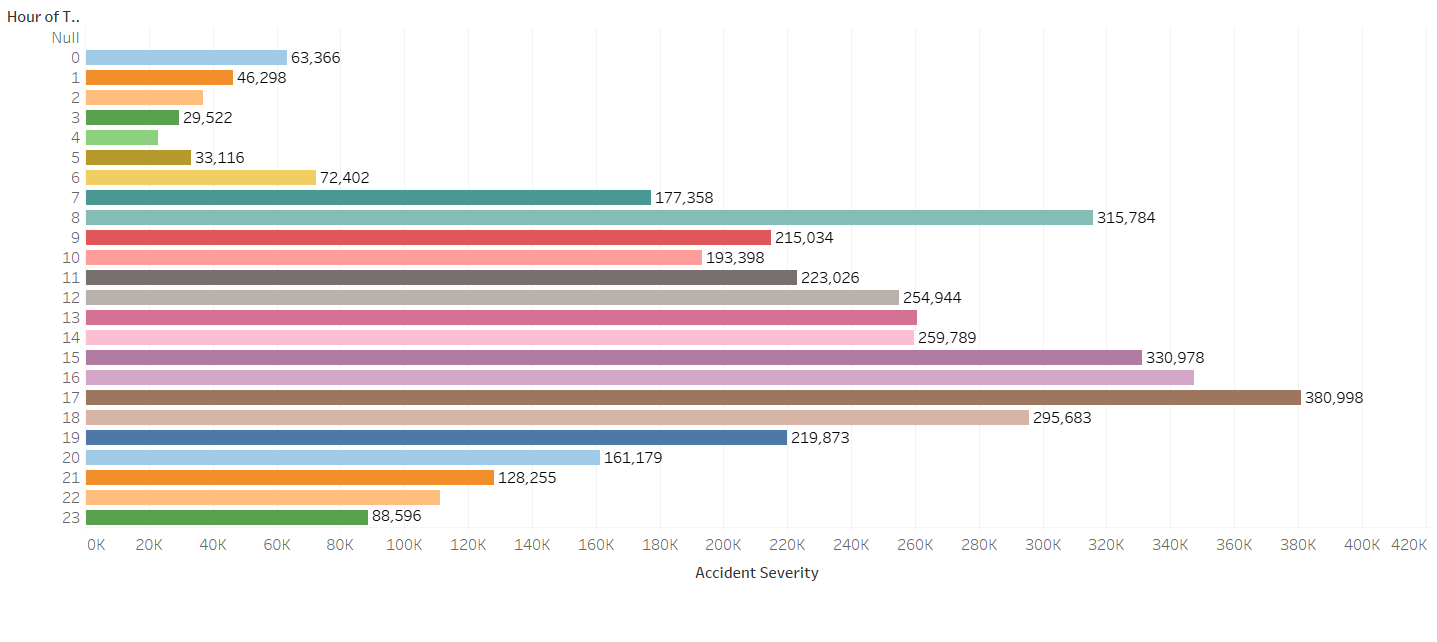
We’ve started with the year, we will go on to quarters, months, Day of the week, Hour of the day.

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

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.

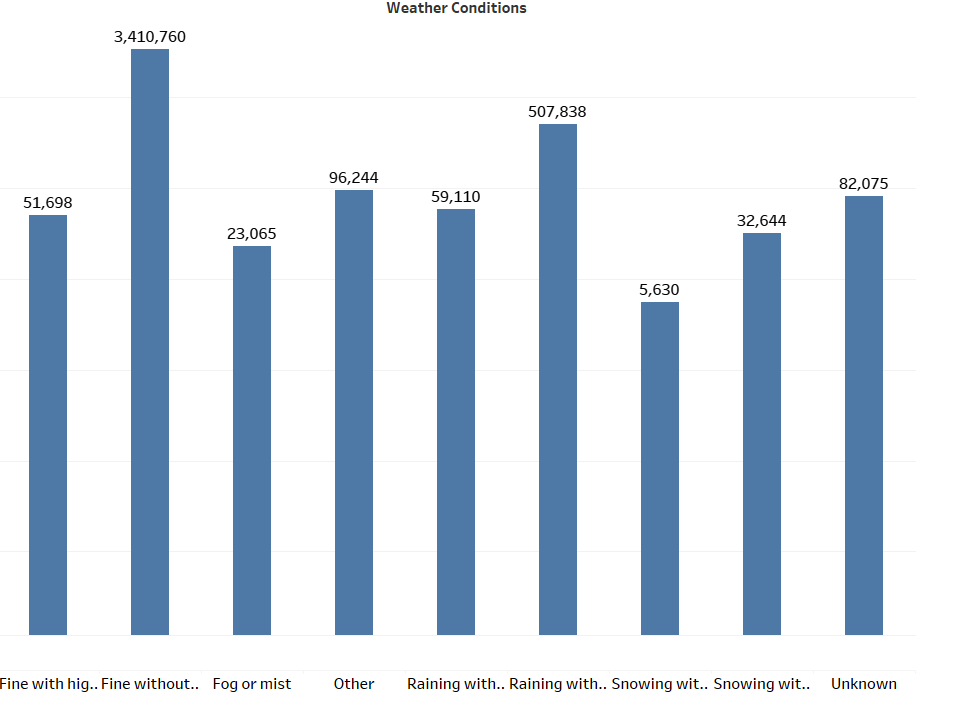
**INDICATING FACTORS**

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.

Figure 5Accident severity according to weather conditions



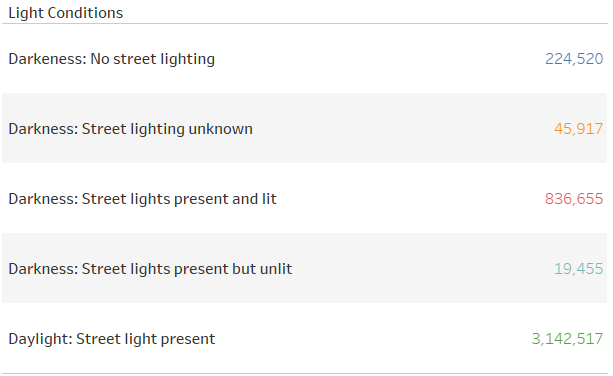
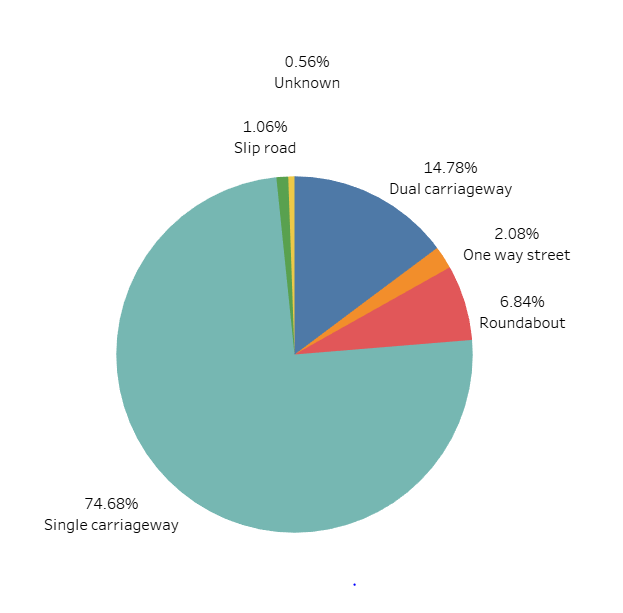


Figure 6: Accident severity according to Light Conditions

As seen from our analysis above, we can barely notice any indications from the dataset because the conditions are pretty normal, in the weather conditions frame, we recoded the highest severity on a *Fine day Without Rain* because on this day, most people are outside, the next in line was raining without winds. For our light conditions, we recorded most accidents in the presence of daylight and even at night when we recorded the most accidents, street lights were present and lit also.



In our pie chart, we can see that the highest severity of accident happens on a single carriageway. The accident severity is really skewed towards the single carriageway which caries about 75% of the total accident severity.

If we consider the fact that this road accommodates two cars coming in the opposite directions at the same time, we can say this is an indication to the fact that accidents are bound to be more severe on this road than on other roads. That is 3 out of 4 of the most severe accidents will take place on a road like this. More of this analysis will be added to the appendix page which can help us fully understand our data set.

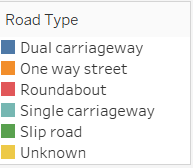


Figure : Accident Severity according to Road Type

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 that 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()

We converted the string columns to numeric with string indexer in pyspark as seen above and afterwards we converted to vector columns as shown below.

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

Now we have the vector columns as shown below

1. df\_vector.show()

CORRELATION MATRIX

To get our correlation matrix, we run this code below

1. # matrix = Correlation.corr(df\_vector, vector\_col)
2. matrix.collect()[0]["pearson({})".format(vector\_col)].values

We have our correlation matrix displayed as an array, we can see that some of the variables have little or no effect on our data set

1. array([ 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. ])

In our correlation matrix, below we can see that we have quite some variables that are not important or let’s say has little or no effect on the Accident Severity. Our other variables are not having the expected effect on the Accident Severity as we would have loved it to indicate but it is quite a mention that we still have some degree of correlation with some of the variables although not strong enough as mentioned earlier but definitely not negative.

We can also take a look at some of the other variables that is showing a stronger degree of correlation, we can easily deduce that the number of casualties and the light conditions are dependent on each other with a fairly strong relationship, which is in fact the strongest of the whole dataset.

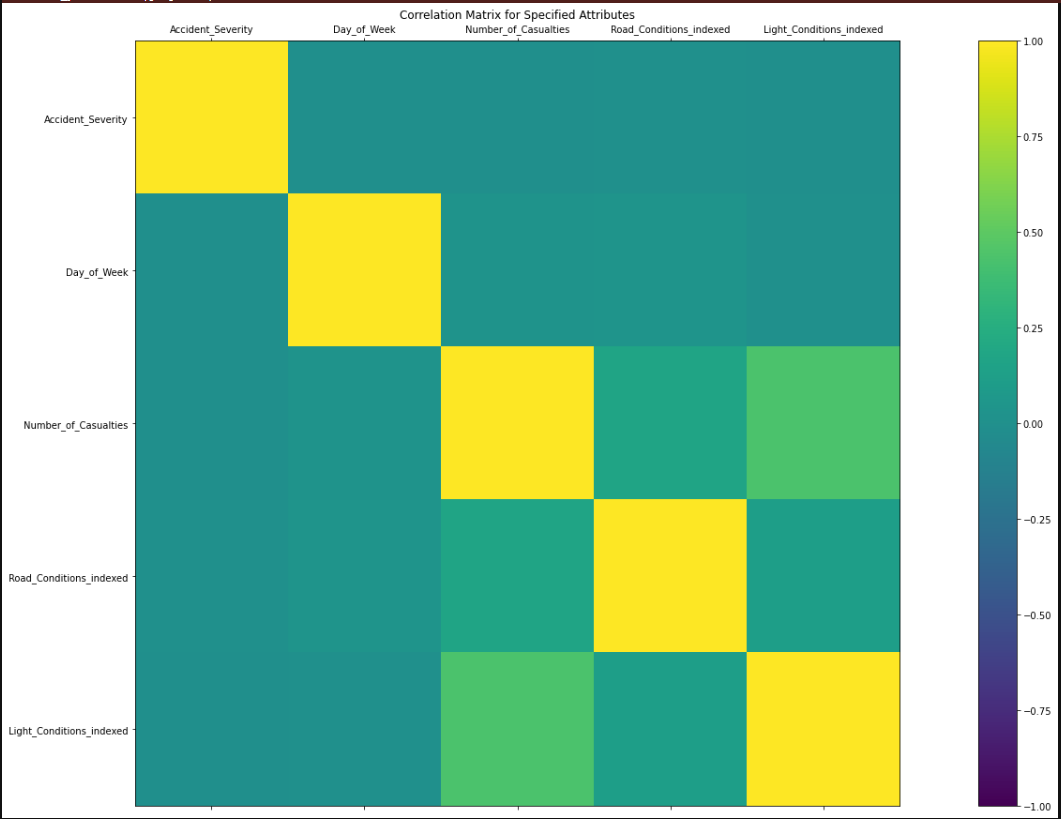


Figure : Correlation Matrix

Let’s perform descriptive analysis on our Dataset

1. from pyspark.ml.linalg import Vectors
2. from pyspark.ml.stat import ChiSquareTest
4. r = ChiSquareTest.test(df\_vector, "features", "Accident\_Severity").head()
6. print("pValues: " + str(r.pValues))
7. print("degreesOfFreedom: " + str(r.degreesOfFreedom))
8. print("statistics: " + str(r.statistics))

pValues: [0.0,0.0,0.0,0.0,0.0,0.0,0.0]

1. degreesOfFreedom: [12, 92, 10, 8, 16, 10, 52]
2. statistics: [2639.459795097584,14299.744114717878,522.2762898287489,10446.739170456356,1696.9419422575206,4986.629696888286,25126.481193806536]

In the frame above we have our test results and our Pvalues came in all through with 0 which means we are rejecting the hypothesis.

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()

We are training a model to predict and also to test our predictions, and see to what level of degree our predictions are correct. The output is attached to this document appendix.

1. from pyspark.ml.evaluation import RegressionEvaluator
2. # create Regressor evaluator object for measuring accuracy
3. evaluator=RegressionEvaluator(metricName='rmse',predictionCol='prediction',labelCol='Accident\_Severity')
4. # apply the RE on predictions dataframe to calculate RMSE
5. rmse=evaluator.evaluate(predicted\_ratings)
6. # print RMSE error
7. print(rmse)

Our output came with a response of ***1.564905762320358*** which shows our model is not a good fit for our dataset which also proves the correlation matrix of little or no strong relationship between the variables and accident severity.

LOGISTIC REGRESSION MODEL

1. from pyspark.ml.classification import LogisticRegression
2. training\_df,test\_df= df\_vector.randomSplit([0.75,0.25])
3. #Apply the logistic regression model
4. log\_reg=LogisticRegression(labelCol='Accident\_Severity').fit(training\_df)
5. #Training Results
6. train\_results=log\_reg.evaluate(training\_df).predictions
7. train\_results.filter(train\_results['Accident\_Severity']==1).filter(train\_results['prediction']==1).select(['Accident\_Severity','prediction','probability']).show(10,False)

The code above trains a logistic regression model and we will try to see if this model is a good fit for our data set. The output of this model is in the source code attached to the appendix page.

Let’s evaluate our model and see if it is a good fit for our data.

1. print('Accuracy: ', MulticlassClassificationEvaluator(labelCol='Accident\_Severity',metricName='accuracy').evaluate(train\_results))
2. print('Precision: ',MulticlassClassificationEvaluator(labelCol='Accident\_Severity',metricName='weightedPrecision').evaluate(train\_results))

Our evaluation came back with the results below.

1. Accuracy: 0.8535078459003081 Precision: 0.7602172790517319

Our results came with a very strong level of precision as shown above, Accuracy of 85% and precision of 76%. We can say this model is a better fit than the previous one

PIPELINE MODEL

In Pyspark, *“A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input DataFrame is transformed as it passes through each stage. For Transformer stages, the transform() method is called on the DataFrame. For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer’s transform() method is called on the DataFrame”* this is fully explained in the official documentation.

1. from pyspark.ml.feature import OneHotEncoder
3. stage\_1 = StringIndexer(inputCol= 'Accident\_Severity', outputCol= 'category\_1\_index')
4. # define stage 2 : transform the column category\_2 to numeric
5. stage\_2 = StringIndexer(inputCol= 'Number\_of\_Casualties', outputCol= 'category\_2\_index')
6. # define stage 3 : one hot encode the numeric category\_2 column
7. stage\_3 = OneHotEncoder(inputCols=['category\_2\_index'], outputCols=['category\_2\_OHE'])
9. # setup the pipeline
10. pipeline = Pipeline(stages=[stage\_1, stage\_2, stage\_3])
12. # fit the pipeline model and transform the data as defined
13. pipeline\_model = pipeline.fit(fin\_df)
14. sample\_df\_updated = pipeline\_model.transform(fin\_df)
16. # view the transformed data
17. sample\_df\_updated.show()

Discussion

Accidents in the UK is on decline from the year 2005 till now, as indicated in our dataset. However, accidents occur in the UK daily. The data analysed in this paper has given us an insight into what needs to be done and actions to be taken. The Single Carriageway, holds for more accidents than any other road, also rush hours during the day (Close and start of work) accounts for the highest time where accidents are due to occur alongside roads with high number of vehicles.

CONCLUSION

Just like any module or framework in python, PYSPARK is easy to install and use. The syntax is quite available to be used even across other languages like R, SCALA, JAVA. The analysis and machine learning process in general is quite straightforward and it leaves the user with much power in the hands, even to train the dataset in a pipeline or in any model as available. Also, transitioning from PANDAS and learning to use a different syntax to what was obtainable in Pandas was quite a task but the online documentation website for PYSPARk provided the needed help and made the transitioning easy to go through.

REFERENCES

Pyspark Apache team. (2021) Pyspark Documentation (Getting Started) [online] Available

from: [PySpark Documentation — PySpark 3.1.1 documentation (apache.org)](https://spark.apache.org/docs/3.1.1/api/python/index.html)

NNK,Admin. Spark with Python (PySpark) Tutorial For Beginners [online] Available

from: [PySpark Tutorial For Beginners | Python Examples - Spark by {Examples} (sparkbyexamples.com)](https://sparkbyexamples.com/pyspark-tutorial/)

Pansen (2021) Correlation Matrix for Pyspark Dataset [online] Available

from: [apache spark - How to get the correlation matrix of a pyspark data frame? - Stack Overflow](https://stackoverflow.com/questions/52214404/how-to-get-the-correlation-matrix-of-a-pyspark-data-frame)

CodersArts (2021) Logistic Regression with Pyspark [online] Available

from: [Logistic Regression Assignment Help using Pyspark (codersarts.com)](https://www.codersarts.com/post/logistic-regression-with-pyspark)

Pyspark Apache team. (2021) Chisquare Test with Python for Spark [online] Available

from: [Basic Statistics - Spark 3.3.0 Documentation (apache.org)](https://spark.apache.org/docs/latest/ml-statistics.html)

Tien Tran (2020) Pyspark for Data Science (Building Models) [online] Available

from: [Pyspark for Data Science | Kaggle](https://www.kaggle.com/code/tientd95/pyspark-for-data-science)

Anonymous. (2020) Pyspark for Model Training and testing [online] Available

from: [PySpark Tutorial for Beginners: Learn with EXAMPLES (guru99.com)](https://www.guru99.com/pyspark-tutorial.html)

[Lakshay Arora](https://www.analyticsvidhya.com/blog/author/lakshayarora/). (2019) Pipelines and model building [online] Available

from: [Building Machine Learning Pipelines using Pyspark (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2019/11/build-machine-learning-pipelines-pyspark/)

APPENDIX

The source code to the programme can be found here [Github](https://github.com/mhoyor/UK-ACCIDENT-ANALYSIS/blob/main/spark.ipynb)

