# CSCE 5300
# INTRODUCTION TO BIG DATA AND DATA SCIENCE

# Road Accident Severity Prediction and Classification using Machine Learning

Group 7:
Prathima Chowdary Sudalagunta
Prathyusha Jampala
Poojitha Pothini
Siva Manikanta Lakumarapu
Rakesh kumar Chekuri

# INDEX


- Abstract
- Introduction
- Requirements
- Dataset
- Data cleaning using pyspark
- Exploratory Data Analysis
- Vector Assembler creation
- Splitting the data
- Classification algorithms
- Regression Algorithms

# ABSTRACT

- Road accidents are the most unwanted thing to a road user,though they happen quite often.
- Due to increasing number of traffic wrecks, it is necessary for us to able to predict the happening of an accident.
- Road accidents can be reduced by accurate prediction based on the factors that cause road accidents.
- The factors that mainly cause road accidents are unfavourable weather conditions,wet/damp roads,vehicle defects and human errors like overspeeding ,alcohol intoxication,etc.

# ABSTRACT


- Our project aims towards to build the efficent model for predicting the accident severity and to classify the accident_severity based on the features present in the dataset using one of the big data technologies pyspark.

# INTRODUCTION

- Road accidents are one of the major causes of mortality, hospitalization and disability.
- The number of fatal road accidents are skyrocketing around the world,killing thousands of people and destroying property everyday without descrimination.
- So,in order to find a solution for the problem, its is important to find out what factors re contributing to the accidents.
- Accident severity analysis invloves three factors namel number of injuries,number of casualities and destruction of property.

# INTRODUCTION

- Severity of an accident is divided into four namely light injury,severe injury, fatal injury and property damage.
- So our project includes determining the causes of the accidents and factors that cause the severity of the accident.
- For this,the phases included are:

➢ Cleaning of data using pySpark.

➢ Seperating the data into training and test datasets.

➢ Clustering features

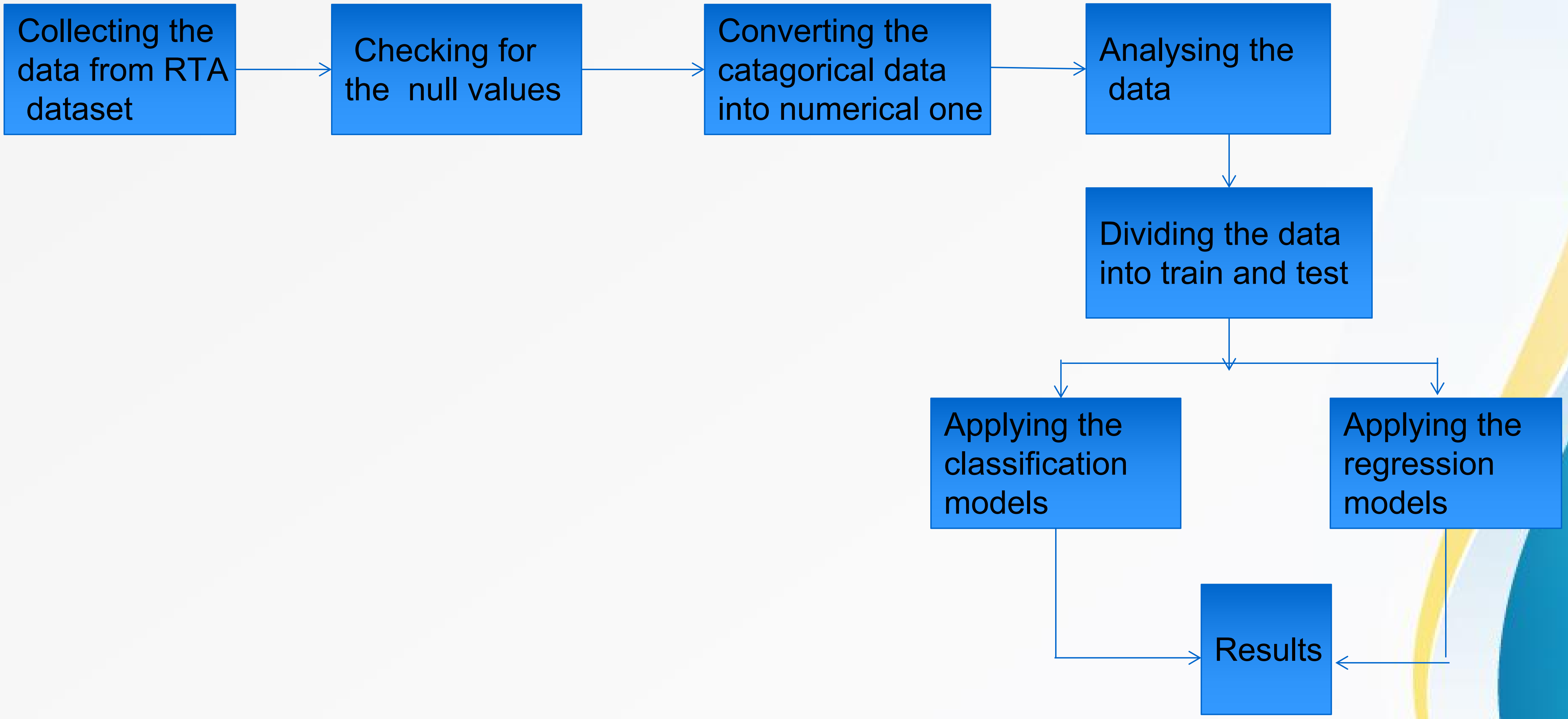
➢ Training automated classifiers

➢ Testing each classifier individually

# Requirements

- Software-Anaconda-jupyter
- Language-Python,Pyspark
- Libraries-Pyspark,Matplotlib,Pandas

## **Work Flow Diagram:**



# DATASET

The dataset we used for this project is Road Traffic accidents by Saurabh Shahane from kaggle and link of our dataset is below:

https://www.kaggle.com/datasets/saurabhshahane/road-traffic-accidents?select=RTA+Dataset.csv

Lets have a look at the head of the data:

Our dataset has 12000 rows and 32 columns as below:

- Time
- Day_of_week
- Age_band_of_driver
- Sex_of_driver
- Educational_level
- Vehicle_driver_relation

# DATASET

- Driving_experience
- Type_of_vehicle
- Owner_of_vehicle
- Service_year_of_vehicle
- Defect_of_vehicle
- Area_accident_occured
- Lanes_or_Medians
- Road_allignment
- Types_of_Junction
- Road_surface_type
- Road_surface_conditions
- Light_conditions
- Weather_conditions

# DATASET

- Type_of_collision
- Number_of_vehicles_involved
- Number_of_casualties
- Vehicle_movement
- Casualty_class
- Sex_of_casualty
- Age_band_of_casualty
- Casualty_severity
- Work_of_casuality
- Fitness_of_casuality
- Pedestrian_movement
- Cause_of_accident
- Accident_severity

# DATASET

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tim | Day_c | Age_bar | Sex_of_ | Educatio | Vehicle_ | Driving | Type_of_ | Owner_ | Service_ | Defect_of_v | Area_ | Lanes_ | Road_al | Types_of_Ju | Road_su | Road | Light_c | Weathe | Type_of_col | Num | Num | Vehicle_ | Casua | Sex_ | Age_ | Cas | Work_ | Fitness | F |
| ## | Mond | 18-30 | Male | Above hi | Employe | 1-2yr | Automo | Owner | Above 1 | No defect | Residential a | | Tangent | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 2 | Going s | na | na | na | na | | | N |
| ## | Mond | 31-50 | Male | Junior hi | Employe | Above | Public (> | Owner | 5-10yrs | No defect | Office | Undiv | Tangent | No junction | Asphalt | Dry | Dayligh | Normal | Vehicle with | 2 | 2 | Going s | na | na | na | na | | | N |
| ## | Mond | 18-30 | Male | Junior hi | Employe | 1-2yr | Lorry (41 | Owner | | No defect | Recre | other | | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 2 | Going s | Driver | Mal | 31-5 | 3 | Driver | | N |
| ## | Sunda | 18-30 | Male | Junior hi | Employe | 5-10yr | Public (> | Governmental | | No defect | Office | other | Tangent | Y Shape | Earth roa | Dry | Darkne | Normal | Vehicle with | 2 | 2 | Going s | Pedes | Fem | 18-3 | 3 | Driver | Norma | N |
| ## | Sunda | 18-30 | Male | Junior hi | Employe | 2-5yr | | Owner | 5-10yrs | No defect | Indus | other | Tangent | Y Shape | Asphalt | Dry | Darkne | Normal | Vehicle with | 2 | 2 | Going s | na | na | na | na | | | N |
| ## | Friday | 31-50 | Male | | Unknown | | | | | | | | | Y Shape | | Dry | Dayligh | Normal | Vehicle with | 1 | 1 | U-Turn | Driver | Mal | 31-5 | 3 | Driver | Norma | N |
| ## | Wedn | 18-30 | Male | Junior hi | Employe | 2-5yr | Automo | Owner | | No defect | Reside | Undiv | Tangent | Crossing | | Dry | Dayligh | Normal | Vehicle with | 1 | 1 | Moving | Driver | Fem | 18-3 | 3 | Driver | Norma | N |
| ## | Friday | 18-30 | Male | Junior hi | Employe | 2-5yr | Automo | Governm | Above 1 | No defect | Reside | other | Tangent | Y Shape | Asphalt | Dry | Dayligh | Normal | Vehicle with | 2 | 1 | U-Turn | na | na | na | na | | Norma | N |
| ## | Friday | 18-30 | Male | Junior hi | Employe | Above | Lorry (41 | Owner | 1-2yr | No defect | Indus | other | Tangent | Y Shape | Earth roa | Dry | Dayligh | Normal | Collision wit | 2 | 1 | Going s | Pedes | Mal | Und | 3 | Driver | Norma | C |
| ## | Friday | 18-30 | Male | Junior hi | Employe | 1-2yr | Automo | Owner | 2-5yrs | No defect | Reside | Undiv | Tangent | Y Shape | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 1 | U-Turn | Passe | Mal | 18-3 | 3 | Driver | Norma | N |
| ## | Satur | 18-30 | Male | Above hi | Owner | 1-2yr | Public (1 | Owner | Unknow | No defect | Reside | other | Tangent | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 1 | Turnove | na | na | na | na | | Norma | N |
| ## | Satur | 31-50 | Male | Above hi | Employe | No Lice | Automo | Owner | | No defect | Office | Undiv | Tangent | No junction | Earth roa | Dry | Dayligh | Normal | Collision wit | 2 | 1 | Going s | Driver | Mal | 18-3 | 3 | Driver | Norma | N |
| ## | Thurs | 18-30 | Male | Junior hi | Employe | 1-2yr | Public (> | Owner | 2-5yrs | No defect | Office | Doubl | Escarpm | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 2 | Going s | na | na | na | na | Driver | Norma | N |
| ## | Thurs | 31-50 | Male | Junior hi | Employe | 5-10yr | Lorry (41 | Owner | Above 1 | No defect | Office | other | Tangent | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 2 | Waiting | na | na | na | na | Other | Norma | N |
| ## | Thurs | 31-50 | Male | Junior hi | Employe | Above | Automo | Owner | 1-2yr | No defect | Office | Undiv | Escarpm | No junction | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 2 | Going s | Driver | Fem | 18-3 | 3 | Driver | Norma | N |
| ## | Mond | 18-30 | Female | Junior hi | Employe | Above | Lorry (11 | Owner | | No defect | Office | One w | Tangent | Y Shape | Asphalt | Wet | Darkne | Raining | Collision wit | 2 | 3 | Going s | na | na | na | na | | | N |
| ## | Mond | 18-30 | Male | Junior hi | Owner | 5-10yr | Public (1 | Owner | | No defect | Office | Undiv | Tangent | Y Shape | Asphalt | Wet | Darkne | Raining | Collision wit | 2 | 3 | Going s | na | na | na | na | Driver | Norma | N |
| ## | Mond | 18-30 | Male | Element | Employe | 2-5yr | Public (1 | Owner | | No defect | Office | other | Tangent | Y Shape | Asphalt | Wet | Darkne | Raining | Collision wit | 2 | 3 | Going s | na | na | na | na | Driver | Norma | N |
| ## | Mond | 18-30 | Male | Junior hi | Employe | 2-5yr | | Owner | Above 1 | No defect | Reside | other | Tangent | Y Shape | Asphalt | Wet | Darkne | Raining | Collision wit | 2 | 3 | Moving | na | na | na | na | Unemp | Norma | N |
| ## | Tuesd | 18-30 | Male | Junior hi | Employe | Below | Long lor | Owner | 5-10yrs | No defect | Reside | One w | Tangent | Y Shape | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 1 | Going s | Driver | Mal | Und | 3 | Driver | Norma | N |
| ## | Tuesd | Under 1 | Male | Junior hi | Employe | 2-5yr | Lorry (41 | Owner | | No defect | Other | One w | Tangent | Y Shape | Asphalt | Dry | Dayligh | Normal | Collision wit | 2 | 1 | Going s | na | na | na | na | Driver | Norma | N |
| ## | Thurs | 18-30 | Male | Junior hi | Employe | 2-5yr | Lorry (11 | Owner | 5-10yrs | No defect | Churc | Undiv | Tangent | Crossing | Asphalt | Dry | Dayligh | Normal | Collision wit | 1 | 1 | Moving | Driver | Mal | 18-3 | 3 | Driver | Norma | N |

# Data cleaning using pyspark

- As our data has number of missing values,it is important to eliminate the disturbing noise and fill the missing data using mean for numeric variables and mode for categorical values.
- For this we imported the following from pyspark as below:

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer,VectorAssembler,OneHotEncoder
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
import matplotlib.pyplot as plt
from pyspark.sql.functions import col, count, isnan, when
```

# Data cleaning using pyspark

- After that open the spark session as below:

```
spark = SparkSession \
    .builder \
    .appName("Road accident Prediction and classification") \
    .getOrCreate()
#https://www.nbshare.io/notebook/187478734/How-To-Read-CSV-File-Using-Python-PySpark/
```

- Read the dataset using spark a shown below

```
#Reading the dataset using pyspark
#https://www.nbshare.io/notebook/187478734/How-To-Read-CSV-File-Using-Python-PySpark/
df_road=spark.read.format("csv").option("header","true").load("C:\\Users\\prath\\Downloads\\archivem\\RTA Dataset.csv")
```

- Print the data as shown below:

```
#https://stackoverflow.com/questions/39067505/pyspark-display-a-spark-data-frame-in-a-table-format
df_road.show(vertical=True)
#Displaying the dataset  using spark
```

# Data cleaning using pyspark

- By displaying the data,the first 20 records of the dataset are shown as below:

```
-RECORD 0------------------------------------------
 Time                          | 17:02:00
 Day_of_week                   | Monday
 Age_band_of_driver            | 18-30
 Sex_of_driver                 | Male
 Educational_level             | Above high school
 Vehicle_driver_relation       | Employee
 Driving_experience            | 1-2yr
 Type_of_vehicle               | Automobile
 Owner_of_vehicle              | Owner
 Service_year_of_vehicle       | Above 10yr
 Defect_of_vehicle             | No defect
 Area_accident_occured         | Residential areas
 Lanes_or_Medians              | null
 Road_allignment               | Tangent road with...
 Types_of_Junction             | No junction
 Road_surface_type             | Asphalt roads
 Road_surface_conditions       | Dry
 Light_conditions              | Daylight
```

- It shows the records in a vertical format with name and value of attributes.

# Data cleaning using pyspark

- Now let us select the columns we needed for the project while removing the other columns from the dataset and store the data in new variable severity_data:

```
selected_columns=['Age_band_of_driver',
 'Sex_of_driver',
 'Educational_level',
 'Vehicle_driver_relation',
 'Driving_experience',
 'Lanes_or_Medians',
 'Types_of_Junction',
 'Road_surface_type',
 'Light_conditions',
 'Weather_conditions',
 'Type_of_collision',
 'Vehicle_movement',
 'Pedestrian_movement',
 'Cause_of_accident',
 'Accident_severity']
```

```
severity_data=df.select(*selected_columns)
```

- Now,let us recreate the dataset with only selected columns:

```
#creating the dataframe using that columns we have selected below and displaying the selected data
road_data=df_road.select(*columns_selected)
road_data.show(vertical=True)
```

# Data cleaning using pyspark

- After creating the dataset with selected columns the dataset is a s shown below:

```
-RECORD 0--------------------------------------
 Age_band_of_driver      | 18-30
 Sex_of_driver           | Male
 Educational_level       | Above high school
 Vehicle_driver_relation | Employee
 Driving_experience      | 1-2yr
 Lanes_or_Medians        | null
 Types_of_Junction       | No junction
 Road_surface_type       | Asphalt roads
 Light_conditions        | Daylight
 Weather_conditions      | Normal
 Type_of_collision       | Collision with ro...
 Vehicle_movement        | Going straight
 Pedestrian_movement     | Not a Pedestrian
 Cause_of_accident       | Moving Backward
 Accident_severity       | Slight Injury
-RECORD 1--------------------------------------
 Age_band_of_driver      | 31-50
 Sex_of_driver           | Male
```

- Now,let us check the recreated dataset for null values:

```
#https://sparkbyexamples.com/pyspark/pyspark-find-count-of-null-none-nan-values/#:~:text=In%20PySpark%20DataFrame%20you%20car
#Checking for the null values in the dataset using select method from pyspark
road_data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in road_data.columns]
   ).show(vertical=True)
```

# Data cleaning using pyspark

- The output gives columns that has missing data and the number of missing values as below:

```
-RECORD 0----------------------
 Age_band_of_driver      | 0
 Sex_of_driver           | 0
 Educational_level       | 741
 Vehicle_driver_relation | 579
 Driving_experience      | 829
 Lanes_or_Medians        | 385
 Types_of_Junction       | 887
 Road_surface_type       | 172
 Light_conditions        | 0
 Weather_conditions      | 0
 Type_of_collision       | 155
 Vehicle_movement        | 308
 Pedestrian_movement     | 0
 Cause_of_accident       | 0
 Accident_severity       | 0
```

- Now remove the null values using dropna() method:

```
#Dropping all the null values using the dropna() function
#https://sparkbyexamples.com/pyspark/pyspark-drop-rows-with-null-values/#:~:text=By%20using%20the%20drop(),result%20in%20NULL
road_data=road_data.dropna()
```

# Data cleaning using pyspark

- After dropping null values from the data let us print the data to check again:

```
#Displaying the removal of null values
#https://sparkbyexamples.com/pyspark/pyspark-find-count-of-null-none-nan-values/?expand_article=1
road_data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in road_data.columns]
   ).show(vertical=True)
```

```
-RECORD 0-----------------------
 Age_band_of_driver      | 0
 Sex_of_driver           | 0
 Educational_level       | 0
 Vehicle_driver_relation | 0
 Driving_experience      | 0
 Lanes_or_Medians        | 0
 Types_of_Junction       | 0
 Road_surface_type       | 0
 Light_conditions        | 0
 Weather_conditions      | 0
 Type_of_collision       | 0
 Vehicle_movement        | 0
 Pedestrian_movement     | 0
 Cause_of_accident       | 0
 Accident_severity       | 0
```

# Data cleaning using pyspark

- Now,if we print the schema of severity_data we can see that every value is a string.

```
severity_data.printSchema()
```

```
root
 |-- Age_band_of_driver: string (nullable = true)
 |-- Sex_of_driver: string (nullable = true)
 |-- Educational_level: string (nullable = true)
 |-- Vehicle_driver_relation: string (nullable = true)
 |-- Driving_experience: string (nullable = true)
 |-- Lanes_or_Medians: string (nullable = true)
 |-- Types_of_Junction: string (nullable = true)
 |-- Road_surface_type: string (nullable = true)
 |-- Light_conditions: string (nullable = true)
 |-- Weather_conditions: string (nullable = true)
 |-- Type_of_collision: string (nullable = true)
 |-- Vehicle_movement: string (nullable = true)
 |-- Pedestrian_movement: string (nullable = true)
 |-- Cause_of_accident: string (nullable = true)
 |-- Accident_severity: string (nullable = true)
```

- Now,let us convert the categorical columns into numerical columns by applying StringIndexer() method of pyspark.

# Data cleaning using pyspark

```
cat_columns=['Age_band_of_driver',
 'Sex_of_driver',
 'Educational_level',
 'Vehicle_driver_relation',
 'Driving_experience',
 'Lanes_or_Medians',
 'Types_of_Junction',
 'Road_surface_type',
 'Light_conditions',
 'Weather_conditions',
 'Type_of_collision',
 'Vehicle_movement',
 'Pedestrian_movement',
 'Cause_of_accident',
 'Accident_severity']
```

```
# Apply the StringIndexer stages to the DataFrame
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index", handleInvalid="keep")
            for col in cat_columns]

indexed_data=road_data
for indexer in indexers:
    indexed_data = indexer.fit(indexed_data).transform(indexed_data)
```

```
indexed_data = indexed_data.drop(*cat_columns)
```

# Data Cleaning using pyspark

- After convertingcategorical values of data into numerical values,the data is as shown below:

```
indexed_data.show()
```

```
+---------------------+-----------------+--------------------+-------------------------+---------------------
+--------------------+--------------------+--------------------+-------------------+---------------------+---
-----------------+-------------------+---------------------+--------------------+---------------------+
|Age_band_of_driver_index|Sex_of_driver_index|Educational_level_index|Vehicle_driver_relation_index|Driving_experience_index
|Lanes_or_Medians_index|Types_of_Junction_index|Road_surface_type_index|Light_conditions_index|Weather_conditions_index|Type
_of_collision_index|Vehicle_movement_index|Pedestrian_movement_index|Cause_of_accident_index|Accident_severity_index|
+---------------------+-----------------+--------------------+-------------------------+---------------------
+--------------------+--------------------+--------------------+-------------------+---------------------+---
-----------------+-------------------+---------------------+--------------------+---------------------+
|                  1.0|              0.0|                 0.0|                      0.0|                  2.0
|                 1.0|                 1.0|                 0.0|                0.0|                  0.0|
0.0|                0.0|                  0.0|                 8.0|                  0.0|
|                  0.0|              0.0|                 0.0|                      0.0|                  3.0
|                 2.0|                 1.0|                 0.0|                0.0|                  0.0|
1.0|                0.0|                  0.0|                 2.0|                  1.0|
|                  0.0|              0.0|                 0.0|                      0.0|                  0.0
|                 2.0|                 0.0|                 1.0|                1.0|                  0.0|
0.0|                0.0|                  0.0|                 1.0|                  0.0|
|                  0.0|              0.0|                 0.0|                      0.0|                  1.0
|                 2.0|                 0.0|                 0.0|                1.0|                  0.0|
0.0|                0.0|                  0.0|                 8.0|                  0.0|
|                  0.0|              0.0|                 0.0|                      0.0|                  1.0
|                 2.0|                 0.0|                 0.0|                0.0|                  0.0|
0.0|               10.0|                  0.0|                 4.0|                  0.0|
|                  0.0|              0.0|                 0.0|                      0.0|                  2.0
|                 2.0|                 0.0|                 1.0|                0.0|                  0.0|
5.0|                0.0|                  3.0|                 1.0|                  0.0|
```

# Exploratory Data Analyzation

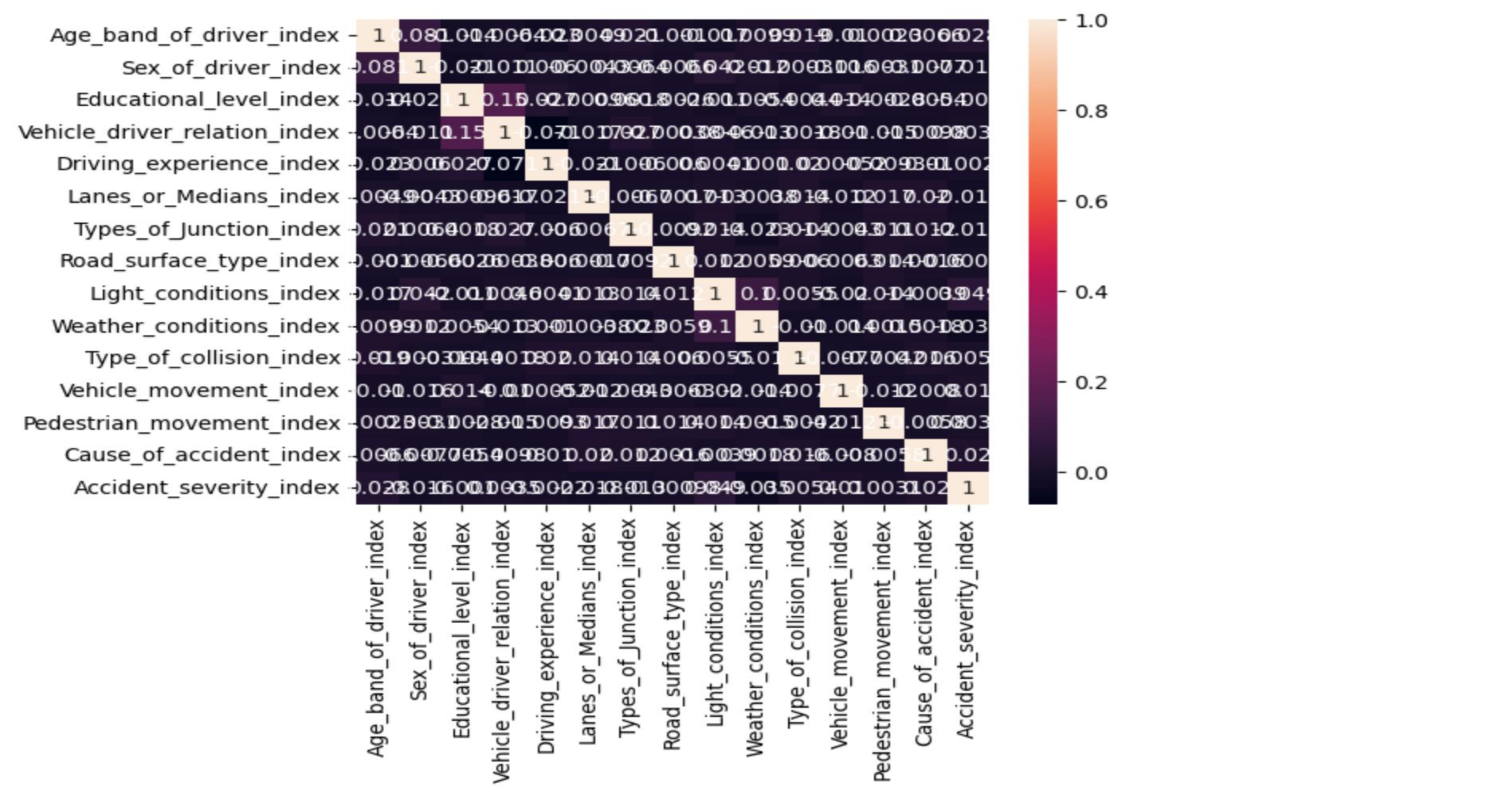
- after data cleaning,le us convert the pyspark dataframe into pandas datafram for data analysis.

```
#converting the pyspark dataframe to pandas dataframe for data analysis
#https://sparkbyexamples.com/pandas/convert-pyspark-dataframe-to-pandas/#:~:text=Convert%20PySpark%20Dataframe%20to%20Pandas%
data2=indexed_data.toPandas()
```

- Let us display the correlation between features in the dataset.

```
#Disolaying the correalation between features in the dataset
sns.heatmap(data2.corr(),annot=True)
plt.show()
```

# Exploratory Data Analyzation



# Exploratory data Analysis

- Let us select the categorical features from dataset for data analysis:

```
#Selecting the features from the dataset for data analysis
categorical_features=['Age_band_of_driver_index',
 'Sex_of_driver_index',
 'Educational_level_index',
 'Vehicle_driver_relation_index',
 'Driving_experience_index',
 'Lanes_or_Medians_index',
 'Types_of_Junction_index',
 'Road_surface_type_index',
 'Light_conditions_index',
 'Weather_conditions_index',
 'Type_of_collision_index',
 'Vehicle_movement_index',
 'Pedestrian_movement_index',
 'Cause_of_accident_index']
```

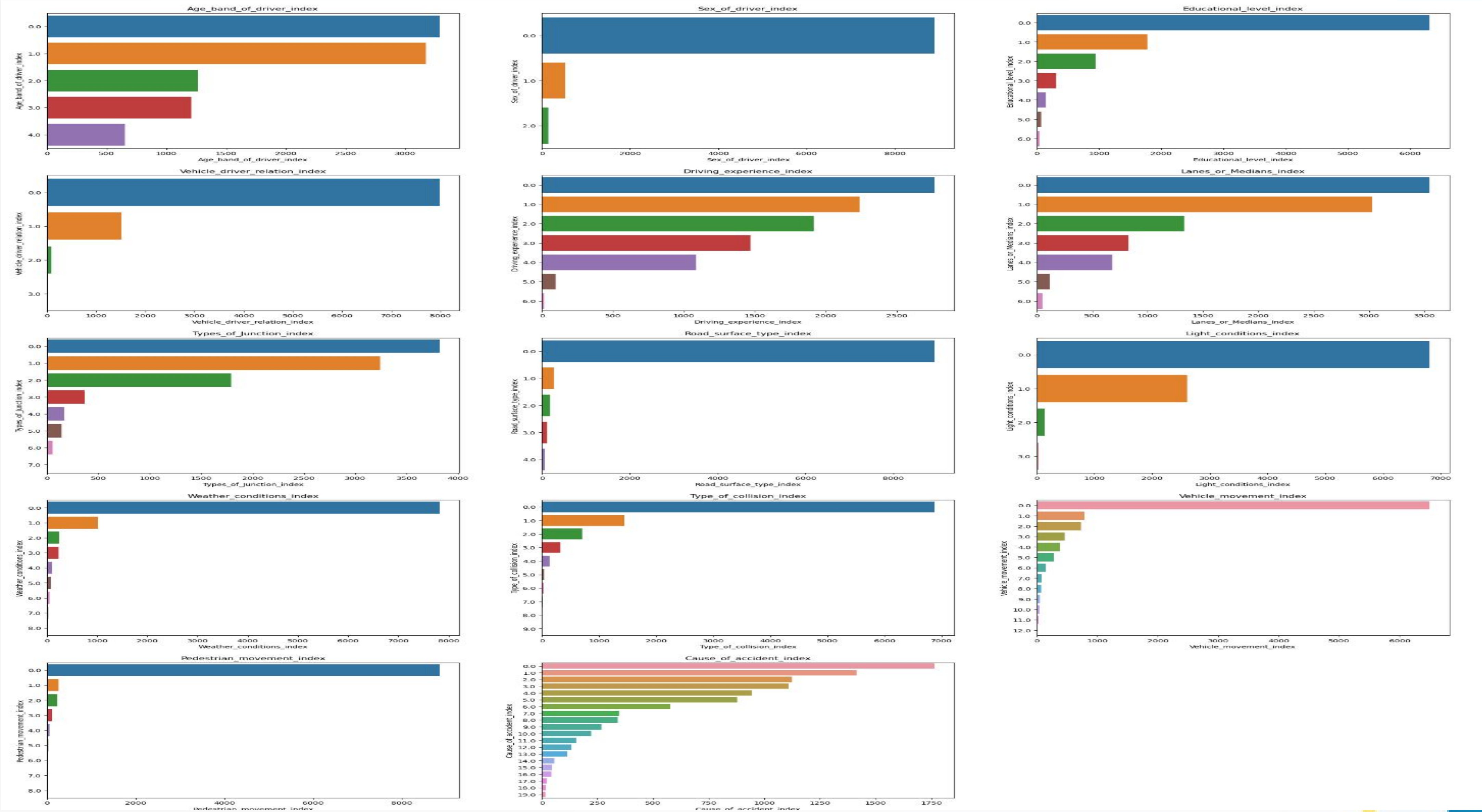
- After selecting categorical features,display the counterplot for the features present in the dataset using seaborn library of pandas.

# Exploratory Data Analysis

The code for displaying counterplot for categorical features is as below:

```
#Displaying the counterplot for the features present in the dataset using seaborn library of pandas
plt.figure(figsize=(30,80), facecolor='white')
plotnumber = 1
for categorical_feature in categorical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.countplot(y=categorical_feature,data=data2)
    plt.xlabel(categorical_feature)
    plt.title(categorical_feature)
    plotnumber+=1
plt.show()
```

# Exploratory Data Analysis



# Creating the Vector Assembler

- Select the features from the dataset and create an assembler for those selected features using VectorAssembler method.

```
#Selecting the features from the dataset
feat_columns=['Age_band_of_driver_index',
 'Sex_of_driver_index',
 'Educational_level_index',
 'Vehicle_driver_relation_index',
 'Driving_experience_index',
 'Lanes_or_Medians_index',
 'Types_of_Junction_index',
 'Road_surface_type_index',
 'Light_conditions_index',
 'Weather_conditions_index',
 'Type_of_collision_index',
 'Vehicle_movement_index',
 'Pedestrian_movement_index',
 'Cause_of_accident_index',
 ]
```

```
#creating an assembler using the VectorAssembler method
assembler = VectorAssembler(inputCols=feat_columns, outputCol="features")
```

# Splitting the Data

- Let us divide the data into traning and testing dataset with 80% of data as training data and 20% of data as testing data.

```
#dividing the dataset into training data and testing data
train_data, test_data = indexed_data.randomSplit([0.8, 0.2], seed=42)
```

- Now let us use assembler to transform the training and testing data as sown below:

```
train_data = assembler.transform(train_data)
test_data = assembler.transform(test_data)
```

# Classificaton Algorithms

- In our project as our data is quite large,we used the following classifiers:

➢ Random Forest Classifier

➢ Logistic regression

➢ Decision Tree classifier

# Random Forest Classifier

- It is a superivsed machine learning algorithm  made up of several random decision trees which work on multiple subsets of the given dataset and provides the prediction accuracy based on the majority votes of predictions from the decision trees.
- Higher the presence of decision trees present in the model, greater accuracy and prevents the problem of overfitting.
- Ensemble learning is the method which combines multiple classifiers and enhance the performance of the model.
- As our data is large and includes multiple classes,it one of the best machine learning algorithm for multi - class classification.

# Random Forest Implementation

```
#https://towardsdatascience.com/a-guide-to-exploit-random-forest-classifier-in-pyspark-46d6999cb5db
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
rf=RandomForestClassifier(featuresCol="features", labelCol="Accident_severity_index",numTrees=50, maxDepth=10)
rf_Model = rf.fit(train_data)
predictions_rf = rf_Model.transform(test_data)
#predictions_rf.collect()
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator_rf_model = MulticlassClassificationEvaluator(labelCol="Accident_severity_index",predictionCol="prediction")
accuracy_rf = evaluator_rf_model.evaluate(predictions_rf)*100
print("Accuracy = %s" % accuracy_rf)
```

```
Accuracy = 78.93853777133887
```

# Logistic Regression

- It is also one of the classification method which is used predict to predict the value of one feature depending on the other.
- It is used to find the relationship between two data features using mathematics.
- There are multiple types of the models in logistic regression like binary,multinomial and ordinal.This multinomial method is used to predict the probability of the dependent variable which has more than possible outcomes.
- As our data contains more than two classes we have used the multinomial method for our classification.
- It also is one of the poweful tools in making the decisions among the classification algorithms.

# Logistic Regression Implementation

```
#https://www.geeksforgeeks.org/logistic-regression-using-pyspark-python/
from pyspark.ml.classification import LogisticRegression
lg= LogisticRegression(featuresCol="features", labelCol="Accident_severity_index",maxIter=1000)
lgModel = lg.fit(train_data)
prediction_log = lgModel.transform(test_data)
evaluator_log = MulticlassClassificationEvaluator(labelCol="Accident_severity_index",predictionCol="prediction")
accuracy_log = evaluator_log.evaluate(prediction_log)*100
print("Accuracy = %s" % accuracy_log)
```

```
Accuracy = 78.83480286427208
```

# Decision Tree classifier

- Decision Tree classifier is also a supervised learning which has some of set of rules to make decision, based on the decisions ,the model will predict the output.
- It is a tree like heirarchical structure contains internal nodes for features in the dataset and branches represent the decision rules,leaf nodes for representing output.
- It is the best systematic approach of classification algorithms of machine learning for classifying the dataset having multiple classes by asking a set opf questions to features of the given dataset.

# Decision Tree classifier

- Moreover, decision tree does not take lots of effort for preprocessing of data and provides accurate results of the inner works like making decisions etc.
- Moreover,this algorithm is the  best when we handling with the large datasets because it will not stop the spped of prediction and doest effect the efficiency of the accuracy.

# Decision Tree implementation

```
#https://www.machinelearningplus.com/pyspark/pyspark-decision-tree/
from pyspark.ml.classification import DecisionTreeClassifier
decision_tree_classifier = DecisionTreeClassifier(featuresCol ='features', labelCol = 'Accident_severity_index')
dtree_model = decision_tree_classifier.fit(train_data)
dtree_predictions = dtree_model.transform(test_data)
#dtree_predictions.collect()
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator_dtree = MulticlassClassificationEvaluator(labelCol="Accident_severity_index", predictionCol="prediction")
accuracy_dtree = evaluator_dtree.evaluate(dtree_predictions)*100
print("Accuracy = %s" % accuracy_dtree)
```

```
Accuracy = 79.09606933104133
```

# Results

| Classification Model | Accuracy |
|---|---|
| Random Forest Classifier | 78.96 |
| Logisitic Regression(multinomial) | 78.83 |
| Decision tree Classifier | 79.02 |

As we can see from the above algorithms, we observe that Random forest and Logistic regression models almost provides the same accuracy but the decision tree classsifier shows some what highest accuracy of 79.02 among the others.

# Regression Algorithms

Regression is a supervised machine learning which is used to map data points to labels.In our project we used the foloowing regression algorithms:

- Random Forest Regressor
- Decision Tree Regressor

# Random Forest Regressor

- It is a superivsed machine learning algorithm made up of several random decision trees which work on multiple subsets of the given dataset and provides the prediction accuracy based on the majority votes of predictions from the decision trees.
- Random Forest regressor is an ensemble learning which combines several decisison trees for accurate prediction.
- Random Forest avoids the problem of overfitting.
- As our data has high dimemsionsionality,we used random forest regressor as it is great with high dimensionality.

# Random Forest Implementation

```
#https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.regression.RandomForestRegressor.html
model = RandomForestRegressor(featuresCol="features", labelCol="Accident_severity_index")  # Replace with the appropriate alg

# Create a pipeline
pipeline = Pipeline(stages=[model])

# Fit the model
pipeline_model = pipeline.fit(train_data)

# Make predictions
predictions_rf_reg = pipeline_model.transform(test_data)

# Evaluate the model
evaluator_rf_reg = RegressionEvaluator(labelCol="Accident_severity_index", metricName="rmse")  # Replace with the appropriate
rmse_rf = evaluator_rf_reg.evaluate(predictions_rf_reg)

# Print the evaluation metric
print("Root Mean Squared Error (RMSE):", rmse_rf)
```

```
Root Mean Squared Error (RMSE): 0.3921869089013273
```

# Decision Tree Regressor

- Decision Tree is also a supervised learning which has some of set of rules to make decision, based on the decisions ,the model will predict the output.
- It is a tree like heirarchical structure contains internal nodes for features in the dataset and branches represent the decision rules,leaf nodes for representing output.
- It observes the features of an object and trains the model in the form of a tree and predicts data in the future.
- We used decisison tree regressor to predict happening of the type of accident.

# Decisison Tree Implementation

```
#ttps://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.regression.DecisionTreeRegressor.html
from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor(featuresCol ='features', labelCol = 'Accident_severity_index')
dt_reg_model = dt.fit(train_data)
dt_reg_predictions = dt_reg_model.transform(test_data)
dt_reg_evaluator = RegressionEvaluator(
    labelCol="Accident_severity_index", predictionCol="prediction", metricName="rmse")
rmse_dt = dt_reg_evaluator.evaluate(dt_reg_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse_dt)
```

```
Root Mean Squared Error (RMSE) on test data = 0.39763
```

# Results

| Regression Model | Root Mean Square Error |
|---|---|
| Random Forest Regressor | 0.3921869089013273 |
| Decisison Tree Regressor | 0.39763 |

As we can see from the above algorithms, we observe that Random forest and Decisison Tree has almost  same Root Mean Squared Error but the RandomForest has less Root Mean Squared Error than Decision Tree regressor.

## Future Scope:

- We can further build the web application for this , so that it enhances the user experience for using this project.
- We will also try to use the other machine learning algorithms or try to implement the efficient methods to increase the accuracy of the model.

# THANK YOU